ALAGAPPA UNIVERSITY

[Accredited with 'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle

and Graded as Category–I University by MHRD-UGC]

KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

Master of Computer Applications

31543



INTERNET AND JAVA PROGRAMMING

IV - Semester

ALAGAPPA UNIVERSITY

(Accredited with 'A+' Grade by NAAC (with CGPA: 3.64) in the Third Cycle and Graded as category - I University by MHRD-UGC)
(A State University Established by the Government of Tamilnadu)

KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

MASTER OF COMPUTER APPLICATIONS

Second Year – Fourth Semester

13543 -- Internet and Java Programming

Author:
Dr. P. Prabhu
Assistant Professor in Information Technology
Directorate of Distance Education
Alagappa University,
Karaikudi. 630 003.

Reviewer:
Dr. S. Santhosh Kumar
Assistant Professor
Department Of Computer Science
Alagappa University,
Karaikudi 630 003

SYLLABI-BOOK MAPPING TABLE
Internet and Java Programming

Syllabi	Mapping in Book
BLOCK 1 INTRODUCTION **Basic Internet Concepts:** Connecting to the Internet – Domain Name System – E-Mail **The World Wide Web** Internet Search Engines – Web Browsers – Chatting and conferencing on the Internet **Online Chatting** Messaging – Usenet Newsgroup – Internet Relay chat (IRC) – FTP – Telnet.	**Pages 1 - 28**
BLOCK 2	
FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING **Basic concepts of OOP** – Benefits – Applications . Java Evolution: Features – how java differs from C and C++ - java and internet- java support system – java environment. Overview of Java Language –Introduction – Simple Java Program – Comments – Java Program Structure – Tokens – Java Statements – Implementing a Java Program – JVM – Command Line Arguments. Constants – Variables – Data Types – Type Casting. Operators and Expressions: Arithmetic Operators – Relational, Logical, Assignment, Increment and Decrement, Conditional, Bitwise, Special Operators – Arithmetic Expressions, Evaluation of Expression – Precedence of Arithmetic Operators – Type Conversions – Operator Precedence and associativity – Mathematical Functions. Decision Making and Branching: If –if…..else –Nesting of if……. Else – else if– switch. Decision Making and Looping: While – do – for – jump in loops – labeled loops.	**Pages 29 – 70**
BLOCK 3 CLASSES, OBJECTS AND METHODS **Class:** Defining a class –fields –methods –creating objects – accessing class members – constructors – methods overloading –static members –nesting of methods – Inheritance –overriding methods –final variables- classes –methods	**Pages 71–108**

CONTENTS PAGE NUMBER

BLOCK 3 : CLASSES, OBJECTS AND METHODS

BLOCK – I INTRODUCTION

UNIT 1
BASIC INTERNET CONCEPTS

Structure

1. 1 Introduction

During the first two decades, computer systems were highly centralized, usually within a single large room called computer centre. The concept of the computer centre as room with a large computer to which users bring their work for processing is rapidly becoming obsolete. A single computer serving all of the organizations computational needs is rapidly being replace in which a large number of separate but interconnected computers do the job. The systems are called computer networks.

Computer network is a set of interconnected autonomous computers to communicate each other. Network usually fall into one of two groups: Local Area Networks (LAN) and Wide Area Networks (WAN). Local area networks connect computers located near each other. Wide Area Network consists of computers in different cities, states or even countries. Personal Area Network (PAN) is a network which enables communication between computer devices near a person. Virtual Private Network (VPN) is a network technology that is used to create a

safe and encrypted connection over a less secure network such as internet. Storage Area Network (SAN) is a block base storage. It is a technology by which high speed architecture connects with servers to their logical disk units.

Internet is a network of network connected to World Wide Web (WWW).Internet gives people the opportunity to connect with anyone in the world at any given time and allows business to be conducted more efficiently and on a grander scale. In essence, the internet makes the world much more efficient. Internet uses the standard Internet Protocol (TCP/IP).Every computer in internet is identified by a unique IP address.IP Address is a unique set of numbers (such as 110.22.31.124) which identifies a computer location.

The first workable prototype of the Internet came in the late 1960s with the creation of ARPANET, or the Advanced Research Projects Agency Network. ... ARPANET adopted TCP/IP on January 1, 1983, and from there researchers began to assemble the “network of networks” that became the modern Internet.Ten years of research brought Local Area Ethernet Networks (LANs) and workstations were developed to get connected to LAN.Computers connected to ARPANET used a standard or rule to communicate with each other with NCP (National Control Protocol).Protocol is a network term used to indicate the standard used by a network for communication.Rapid change in information technology suppressed NCP and brought TCP/IP (Transmission Control Protocol/Internet Protocol) in to the world of networking

Need for Internet:

Most of us use, the internet as a way to connect with other people, sharing information, sharing of files, for entertainment, socializing, and many other things that could be beneficial for us.The internet is necessary for the following reasons;

Communication – People use the Internet to communicate with one another using e-mail. Software has made it possible to stream voice and video across the world with minimal delay, and email has become the main means of communicating for many a modern person.

Entertainment – Many people use the Internet to enjoy themselves and to engage in personal interests. In recent years, multiple player games and virtual worlds have engaged the time and money of many. Plus, video and music are easy to find, stream and download.

Market – People use the Internet to research, find and buy services and products or to target and sell to the ultimate consumer. In effect, the Internet has become THE best way to buy and sell merchandise, as online "stores" are open 24 hours a day, 7 days a week.

Ask for Help - Some people use the Internet to ask for help. People ask for help in the form of emotional support, medical advice, or even simply listening.

Education and Research - People use internet for education to learn and research purpose

Electronic Newspapers - People use electronic news papers and magazines to know the current affairs.

Relationships& Discussion Groups – People use the Internet to find, maintain, or end relationships. But people can get addicted to social networks, too.

The other needs of internet include downloading files, utilizing services through cloud computing job hunting real time applications such as railway reservation and online shopping.Internet allows us to use many services like:

- Internet Banking
- Matrimonial Services
- Online Ticket Booking
- Online Bill Payment
- Data Sharing
- E-mail

In this unit you will learn about the structure of DNS, Name Server concept and resolver, e-mail communication and world wide web..

1.2 Objectives

After going through the unit you will be able to;

- Understand the fundamentals of computer networks
- Know about the Domain Name System
- Discuss about e-mail communication system
- Learn how to connect to the Internet

1.3 Connecting to the Internet

When connecting one or more computers for communication, it forms a network. We can also connect two or more networks and form an internetwork or internet.

An Internet service provider (ISP) is an organization that provides services for accessing, using, or participating in the Internet. Internet services typically provided by ISPs include Internet access, Internet transit, domain name registration, web hosting, Usenet service, and colocation. Internet service providers (ISPs) offering broadband and narrow band internet services in India. They are namely Jio,Airtel,Vodaphone,Idea Cellular, BSNL,Tata Teleservices, ACT,MTNL and Hathway.

The type of Internet service you choose will largely depend on which Internet service providers (ISPs) serve your area, along with the types of service they offer. Here are some common types of Internet service.

Dial-up:

This is generally the slowest type of Internet connection, and you should probably avoid it unless it is the only service available in your area. Dial-up Internet uses your phone line, so unless you have multiple phone lines you will not be able to use your landline and the Internet at the same time.

Leased Connection

Leased connection is also known as direct Internet access or LevelThree connection. It is the secure, dedicated and most expensive,level of Internet connection. With leased connection, your computeris dedicatedly and directly connected to the Internet using highspeed transmission lines. It is on-line twenty-four hours a day, sevendays a week. Leased Internet connections are limited to large corporations and universities who could afford the cost.

Digital Subscriber Line (DSL):

Digital Subscriber Line (DSL) is the newest technology being usedfor Internet access. DSL connects your home or office to the Internetthrough the same telephone wire that comes from the telephonepole on the street. Like ISDN, with DSL, user can make and receivetelephone calls while connected simultaneously to the Internet.However, DSL service is limited

in the distance that you can be fromthe provider’s point of presence (POP).To use DSL you will need a DSL modem (also called a router), aNetwork Interface Card (NIC), and a telephone line. DSL is also relatively new technology that is just being introduced in many places.Hardware developers are working with service providers to make theservice cost affordable for consumers. As time goes by, the serviceshould become more widely available at a reasonable price.

Cable:

Cable service connects to the Internet via cable TV, although you do not necessarily need to have cable TV in order to get it. It uses a broadband connection and can be faster than both dial-up and DSL service; however, it is only available where cable TV is available.

Satellite:

A satellite connection uses broadband but does not require cable or phone lines; it connects to the Internet through satellites orbiting the Earth. As a result, it can be used almost anywhere in the world, but the connection may be affected by weather patterns. Satellite connections are also usually slower than DSL or cable.

3G and 4G:

3G and 4G service is most commonly used with mobile phones, and it connects wirelessly through your ISP's network. However, these types of connections aren't always as fast as DSL or cable. They will also limit the amount of data you can use each month, which isn't the case with most broadband plans.

Modem

Once you have your computer, you really don't need much additional hardware to connect to the Internet. The primary piece of hardware you need is a modem.A modem transmits binary data by representing a 0 as decreased frequency signal and 1 as increased frequency signal. The computer uses a modem to translate binary data into analog signal for transmission over telephone line. The figure 1.1 shows the example model used to connect Internet.

NOTES



Figure 1.1A Modem

Router

A router is a hardware device that allows you to connect several computers and other devices to a single Internet connection, which is known as a home network. Many routers are wireless, which allows you to create a home wireless network, commonly known as a Wi-Fi network.



Figure 1.2 Router

Web Browser

A web browser, or simply "browser," is a Software / application used to access and view websites. Common web browsers include Microsoft Internet Explorer, Google Chrome, Mozilla Firefox, and Apple Safari.



1.3 Most Popular Web Browsers

The primary function of a web browser is to render HTML, the code used to design or "mark up" webpages.Browsers' benefits and key features include;

- They're free to download.
- You can have more than one on your computer.
- They all work in a similar way.
- They allow users to explore websites anywhere on the internet.
- Can be personalized by allowing users to add favorites or set a different home page (the first page that you see when you open your browser).

Search Engine:

A search engine is a web-based tool that enables users to locate information on the World Wide Web. Generally, people use search engines for one of three things: research, shopping, or entertainment. Most people who are using a search engine are doing it for research purposes. They are generally looking for answers or at least to data with which to make a decision. Types of Search Engines include;

- Crawler Based Search Engines.
- Directories Search Engines.
- Hybrid Search Engines.
- Meta Search Engines

Popular examples of search engines are Google, Yahoo!, ask,altavista and MSN Search.



1.4 Most Popular Search engines

1.4 Domain Name System (DNS)

Network computer users generally prefer to use domain names instead of dotted decimal addresses. Internet programs need to use 32 bit IP addresses. As such to ensure successful internetwork communications between the programs and users we need fast, reliable method of translating between address schemes. For this reason, network engineers developed name-server software. A name server is a program that translates domain name into IP addresses.

Domain Name System DNS is a client/ server based distributed database system. The DNs distributes specific address details among various name servers.

The internet Domain Name system uses names like ftp.microsoft.com to identify specific computer.Each element is referred as a label. For example ftp.microsoft.com consists of three label namely ftp, microsoft and com. Your separate internet name labels with a period that you read as a word dot. In other words, you would say the name ftp.microsoft.com as ftp dot Microsoft dot com. Domain is defined as a sphere of activity, concern or function. The label Microsoft describes the organisation or entity the owns the compute Microsoft corporation. The label com represents organisation uses the computer for commercial enterprises.

The following figure 1.5 shows the hierarchy of Domain name system. It can be viewed as organization chart. The top of the chart is an unnamed starting point called root. The DNS root is like directory on the disk – neither root has a name. However, like directories on the computer, each domain has the name. Just each directory is divided into sub-directories; the Internet Domain further divides each domain into sub-domains.

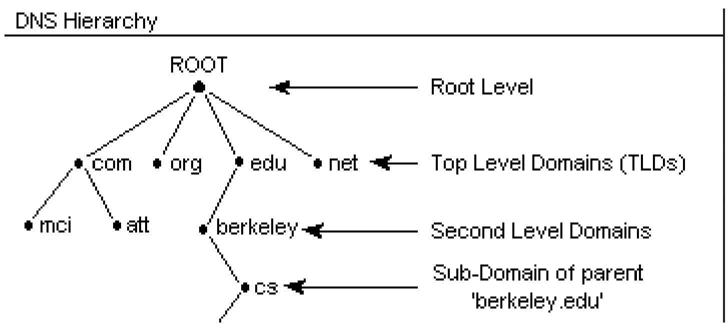


Figure 1.5 The hierarchical structure of Internet Domain Name System

The level immediately below the root of the Internet DNS consists of three top domains. The internet usually divides this group of domains into seven basic categories. The given table identifies the seven basic classifications of the organizational domains.

Domain	Description
com	Commercial organizations such as business
edu	Educational organizations such as universities
gov	US government organizations etc
Int	International organizations
mil	US military organization
net	A network that doesn't fit into one of the other organizational domain categories
org	An organization that doesn't fit into one of the other organizational domain categories

Uniform Resource Locators (URL)

An URL identifies a resource, normally a file. It is the computer address of a resource, such as web document, a file or a program. An example of url is https://www.alagappauniversity.ac.in/. When typing the URL into your browsers location field and hit enter, your browser will attempt to connect you to the resource at that URL address. After you connect to a web server, your browser displays the web information set to it by the server.

1.5 Electronic Mail – E-mail

Electronic mail is the ability to send and receive messages via computer. On most networks, it is the most widely used application. It is different from the other network applications. It is a store-and=forward service that works very much like regular postal service.Features of e-mail include;

- One-to-one or one-to-many communications
- Instant communications
- Physical presence of recipient is not required
- Most inexpensive mail services, 24-hours a day and seven days a week
- Encourages informal communications

Components of an E-mail Address

As in the case of normal mail system, e-mail is also based upon theconcept of a recipient address. The email address provides all of theinformation required to get a message to the recipient from anywhere in the world. Consider the e-mail ID

dde@alagappauniversity.ac.in

In the example above, “dde” is the local part, which is the name of a mailbox on the destination computer, where finally the mail will be delivered. alagappauniversity is the mail server where the mailbox “dde” exist, .com is the type of organization on net, which is hosting the mail server.There are six main categories;

- com Commercial institutions or organization
- edu Educational institutions
- gov Government site
- mil Military site
- net Gateways and administrative hosts
- org Private organizations

E-mail lets people communicate synchronously; you can send mail messages whenever you want, and the people who receive mail from you can read the message whenever they want.

When you send e-mail, your message is forwarded from one computer to another until destination is reached. At the destination, your message goes to the recipient’s system mail box, a file that holds the users incoming messages.

The following figure 1.6 shows the basic elements of e-mail system.

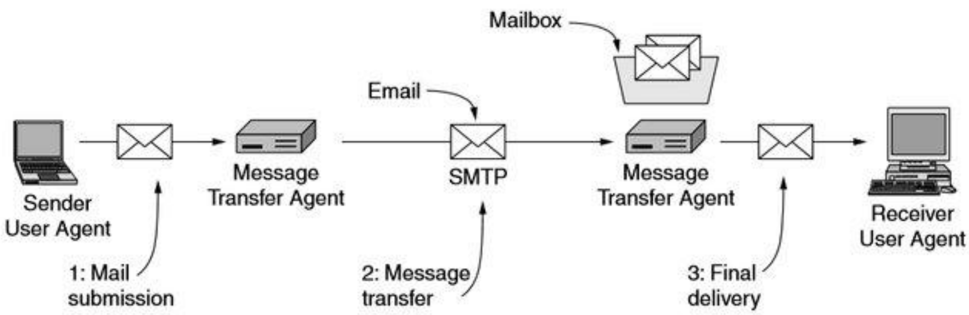


Figure 1.6 Components of e-mail system

Each e-mail message has a sender and receiver, both of which have a user interface into the Internet electronic mail system.

E-mail Message Components

E-mail message comprises of different components: E-mail Header, Greeting, Text, and Signature. These components are described in the following figure 1.7

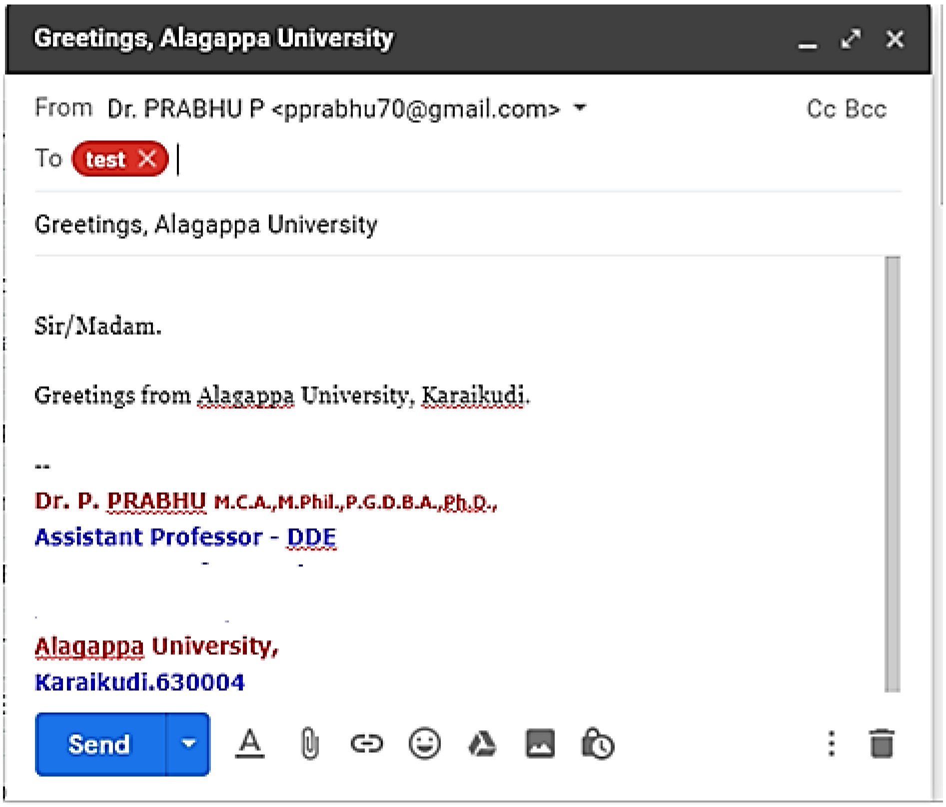


Figure 1.7 e-mail components

E-mail Header

The first five lines of an E-mail message is called E-mail header. The header part comprises of following fields:

- From
- Date
- To

- Subject
- CC
- BCC

From

The From field indicates the sender's address i.e. who sent the e-mail.

Date

The Date field indicates the date when the e-mail was sent.

To

The To field indicates the recipient's address i.e. to whom the e-mail is sent.

Subject

The Subject field indicates the purpose of e-mail. It should be precise and to the point.

CC

CC stands for Carbon copy. It includes those recipient addresses whom we want to keep informed but not exactly the intended recipient.

BCC

BCC stands for Black Carbon Copy. It is used when we do not want one or more of the recipients to know that someone else was copied on the message.

Greeting

Greeting is the opening of the actual message. Eg. Hi Sir or Hi Guys etc.

Text

It represents the actual content of the message.

Signature

This is the final part of an e-mail message. It includes Name of Sender, Address, and Contact Number.

Advantages

E-mail has proved to be powerful and reliable medium of communication. Here are the benefits of E-mail:

- Reliable
- Convenience
- Speed
- Inexpensive
- Printable
- Global

Disadvantages

Apart from several benefits of E-mail, there also exists some disadvantages as discussed below:

- Forgery
- Overload
- Misdirection
- Junk
- No response

The Internet e-mail system consists of an outgoing queue, a client process, a server process and mailboxes for incoming mail. A mail box can refer to user address or a container file that stores e-mail data. The user agent replaces the e-mail program and the message transfer agent replaces the client and server process. Most Internet e-mail specifications refer to an e-mail program as a user agent. Likewise, it message transfer agent is a client or server program that performs e-mail related services such as sending and receiving mail for a host computer. You interact with user agent program which in turn interacts with an email container on your behalf. At the same time, the MTA program acts as an agent on behalf of a host computer. The user agent shields your from interacting with a wide variety of user agents or other MTA's.

Conceptually, the user agent to an email system is separate from the message transfer agent. Although you can implement both the user agent and message transfer agent in a single program. You should isolate the design of each agent in separate modules. The agents perform

very different functions through they are closely related. On the Internet, the message transfer agent represents the internet email. The MTA's that establish TCP connections to communicate with other MTAs typically use the Simple Mail Transfer Protocol (SMTP)

Simple Mail Transfer Protocol (SMTP)

The core of the internet's e-mail system is the message transfer agent. The message transfer agent represents the e-mail system to a host computer. Although internet e-mail users rarely work with a message transfer agent MTAs play a crucial role in all e-mail transmissions.

SMTP is similar to FTP in many ways, including the same simplicity of operation. SMTP uses TCP port number 25. It uses spools or queues. When a message is sent to SMTP, it places it in a queue. It attempts to forward the message from the queue whenever it connects to remote machines.

1.6 Check Your Progress

1. What is Domain Name Server? Explain its structure.
2. List some organizational domains and its description.
3. Name some of Internet Service Providers (ISP) in India.

1.7 Answers to Check Your Progress Questions

1. The internet Domain Name system uses names like ftp.microsoft.com to identify specific computer.

2. List of domains are

Domain	**Description**
com	Commercial organizations such as business
edu	Educational organizations such as universities
gov	US government organizations etc
Int	International organizations
mil	US military organization
net	A network that doesn't fit into one of the other organizational domain categories
org	An organization that doesn't fit into one of the other organizational domain categories

3. Jio, Airtel, Vodafone, Idea Cellular, BSNL, Tata Teleservices, ACT,MTNL and Hathway are some of common Internet Service Providers in India.

1.8 Summary

Computer network is a set of interconnected autonomous computers to communicate each other. When connecting one or more computers for communication, it forms a network. We can also connect two or more networks and form an internetwork or internet. Electronic mail is the ability to send and receive messages via computer.

1.9 Key Words

- **Electronic Mail -** Electronic mail is the ability to send and receive messages via computer. On most networks, it is the most widely used application.

- **Computer network -** It is a set of interconnected autonomous computers to communicate each other

- **Router** A router is a hardware device that allows you to connect several computers and other devices to a single Internet connection, which is known as a home network.

1.10 Self-Assessment Questions and Exercises

1. Describe the functions of e-mail.
2. Explain in detail about Domain Name system with examples.
3. How will you connect with internet? What are the components required to connect Internet?
4. Explain in detail about some common types of Internet service.

1.11 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004

NOTES

2. E. Balagurusamy, Programming with Java, 4e,Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998

UNIT – 2
THE WORLD WIDE WEB (WWW)

Structure

2.1 Introduction

The term World Wide Web (WWW, or W3 or Web) and internet are widely used word in the communication technology. World Wide Web is a collection millions of linked HTML(Hyper Text Markup Language) pages on the internet. We can view text, graphics, video, and hear audio when we use our browser to access web sites. The internet is a vehicle that lets computers around the world communicates. WWW was originally developed at CERN(Conseil European Pour la Recherche Nucleaire), Geneva's European Laboratory for Particle Physics, the high-energy physics research centre in Switzerland. Today more than one million web servers exist in the world. No web has grown dramatically and has taken on a new appearance. The web is no longer limited to scientific information exchange by researchers. Business uses the web to advertise their products.

To surf the web, we must have a computer with a modern and a telephone connection. Also we need browser software such as Netscape navigator or Internet Explorer or Google Chrome. A web browser is used to view the web pages on the internet.WWW clients on theinternet can display pages from any of the nearly 10000 web servers. Each time we choose a link; WWW Connects to the appropriate server retrieves the next page and returns control to the local client. WWW is thus an efficient method of finding and using the information widely dispersed throughout the world.

NOTES

In Internet, Web operations are depending on a protocol called Hyper Text Transfer Protocol (HTTP). Hyper Text Transfer Protocol (HTTP) is used to transfer HTML files between web pages over the World Wide Web. HTML is the mostly widely used language for developing web pages. All web browsers follow internet standards for accessing web pages. World Wide Web is composed of a set of HTML web pages.

2.2 Objectives

After going through the unit you will be able to;

- Understand about the World Wide Web
- Know about web browsers and its usage
- Explain about chatting and conferencing on the internet
- Discuss the functions of internet search engines
- List the names of various search engines
- Name various types of web browsers.

2.3 Internet Search Engines

Search engine is a service that allows Internet users to search for content via the World Wide Web (WWW). A user enters keywords or key phrases into a search engine and receives a list of Web content results in the form of websites, images, videos or other online data. The list of content returned via a search engine to a user is known as a search engine results page (SERP).(Source ;Tecopedia)

For example

- Google.
- Bing.
- Yahoo.
- Ask.com.
- AOL.com.
- Baidu.
- Wolframalpha.
- DuckDuckGo.

2.4 Web Browser

A browser is a software application used to locate, retrieve and display content and languages such as XML or Extensible Markup Language, HTML or Hyper Text Markup Language. on the World Wide Web,

including Web pages, images, video and other files. As a client/server model, the browser is the client run on a computer that contacts the Web server and requests information. There are varieties of web browsers available with different advantages and working on different platforms. The most popular web browsers are;

- Hot Java
- Netscape Navigator
- Avant
- Internet Explorer
- Google Chrome
- Mozilla fire box
- Opera and
- Safari.

Internet Explorer was developed by Microsoft in 1994 and released in 1995. Mozilla fire box browser was developed by Mozilla Corporation in 2004. This is the second most popular browser after Internet Explorer. Safari is the web browser made by apple with compatibility with Microsoft Windows iPhone OS and Mac OS. Opera is a web browser developed by Opera Software specially used for mobile applications. Google Chrome is another web browser developed by Google in September 2008 for Windows. Netscape navigator is also a web browser developed by Netscape communications which supports almost all Operating systems. The following figure 2.1 shows the example internet explorer web

browser.

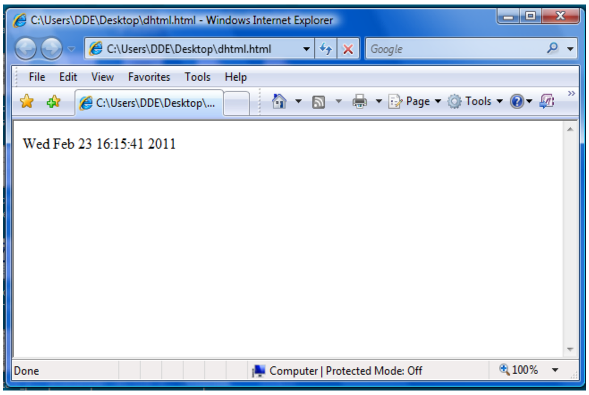


Figure 2.1 Example Web Browser – Internet Explorer

The most distinguish areas of web browsers are platform (Unix, Windows, Mac, Linux), protocols(FTP,SMTP,HTTP,IMAP), GUI (Graphical User Interface), mobile compatibility and open source.

2.5Chatting and Conferencing on the Internet

Internet is used in many areas; some of the areas where the internet is used is as follows;

- e-mail
- website
- e-commerce
- social media
- resource sharing
- Conferencing
- Chatting
- Online courses

2.5.1 Chatting

Chatting is almost as good as speaking to the other person over the phone. Online chatting is a online conversation between users or group of users. There are many free chat sites on the web through which you can communicate with one or many people from different locations. After you log on to free chat site, go to the specific chat room where you have agreed to meet other person. For example:

1. Type the site address www.yahoo.com
2. Click on chat option
3. Click on sign up for yahoo chat to create a new account
4. If you already have an account with yahoo you can enter Yahoo Id and Password and click on Sign In button.
5. Click on Start Chatting button.

2.5.2 Conferencing

Conferencing comprises the technologies for transmission of audio/video signals by users from different places for communication between users on the network. Video conferencing is also called as video collaboration is a type of network groupware. Video conferencing systems during 1990's are expensive communication systems with network and software requirements. Video conferencing using two protocols such as Session Initialization Protocol (SIP) and H3.xxx.

SIP is a protocol for creating, modifying and terminating sessions with one or more users. The architecture of SIP is similar to Hyper Text Markup Protocol(HTTP). The client request services to the server for processing the tasks. The server receives the request and processes it and responds

back to the client. This protocol itself provides reliability, which is not depending on TCP/IP.

The SIP system has two components; user agents and network servers. A user agent is an end system acting on behalf the user. It has two parts namely client and server. There are three types of network servers namely proxy server, registration server and redirect server. A registration server receives updates about the user location. A proxy servers receives the message and forwards to next-hop server. A redirect server receives the message, and determine the next-hop server to the client.

H.3XX are ITU-T study group XVI umbrella recommendations for video conferencing. These recommendations refer other recommendations that include multiplexing, control and signalling.

2.6 Check your Progress

1. What is web browser? Give examples
2. List any two internet search engines.

2.7 Answers to Check Your Progress Questions

1. A browser is a software application used to locate, retrieve and display content and languages such as XML or Extensible Markup Language, HTML or Hyper Text Markup Language. on the World Wide Web, including Web pages, images, video and other files. Examples : Google chrome and Internet Explorer

2. The internet search engines are : google, altavista, yahoo

2.8 Summary

World Wide Web (WWW) is a collection of linked HTML(Hyper Text Markup Language) pages on the internet. Search engine is a service that allows Internet users to search for content via the World Wide Web (WWW). A browser is a software application used to locate, retrieve and display content and languages such as XML or Extensible Markup Language, HTML or Hyper Text Markup Language. on the World Wide Web, including Web pages, images, video and other files.

Conferencing comprises the technologies for transmission of audio/video signals by users from different places for communication between users on the network. Video conferencing is also called as video collaboration is a type of network groupware.

2.9 Key Words

- **Session Initialization Protocol - SIP** is a protocol for creating, modifying and terminating sessions with one or more users.

- **Chatting** is almost as good as speaking to the other person over the phone.

2.10 Self-Assessment Questions and Exercises

1. What do you meant by chatting
2. What are two types of protocols used for video conferencing? Explain
3. Write short note on : web Browser
4. List any two names of web browsers.
5. Explain the purpose of internet search engines? Explain.

2.11 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998.

NOTES

UNIT – 3
ONLINE CHATTING

Structure

3.1 Introduction

Communication plays major role in internet applications. It provides sharing of information, messages between user and/or group of users. There are various applications such as online chatting, messenger and user networks groups provide opportunity to share information over the network.

3.2 Objectives

After going through the unit you will be able to;

- Understand about online chatting
- Know about the message between network user
- Explain about the user network groups
- Discuss about file transfer protocol
- Learn about the use of Telnet

3.3 Online Chatting and Messaging

Chatting is to talk in a way with one person with other or group of persons over the telecommunication especially internet. Also it refers to the way of

NOTES

communicating or interacting with users over internet. It can be established by connecting one user with other users using chat software.

The chat software creates one or more rooms for users to communicate with each other according to the user's interest. To chat with users, one who need internet relay chat or instant messenger application. Chat may be delivered through verbal, audio, text and video.

3.4 Usenet Newsgroup

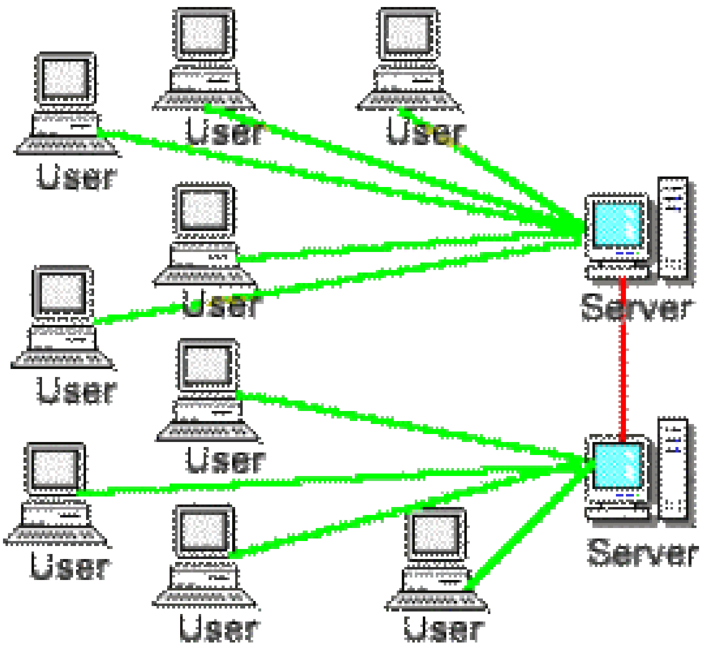
Usenet Newsgroup is storage with in the usenet system where the users from different geographical locations post their messages over internet. The usenet files uploaded by the users are stored in the servers distributed over the various geographical locations and connected through the internet. The sending and receiving of files is performed by the Network News Transfer Protocol (NNTP) which allows connection to Usenet servers and data transfer over the internet. The communication is established between server-server and client-server. Newsgroups are divided into two ways; text and binary which depend on bandwidth.

3.5 Internet Relay chat (IRC)

Internet Relay Chat (IRC) is an application layer protocol which is used to communicate in the form of text. It was developed in August 1988, by Jarkko Oikarinel. IRC is used to enable the users to request to a server using server side programming software and communicate with each other online.

The IRC client software is used to connect to the IRC server. There are some small IRC servers available for Internet Relay chat.The small IRC servers are namely OperaNet which has less number of users. The medium IRC servers are freenode and Dalnet which may have 40,000 users.The big IRC servers may consist of over 100,000 users; for example, EFNet and UnderNet. The example IRC is shown in figure 3.1

NOTES



3.1 Internet Relay Chat System

IRC is a messaging application for chatting which consists of set of rules and software for client/server technology. One can start a chat room(Called a channel) or join an existing chat room. Protocols are used to identify existing chat groups. The nick names can be used to identify particular user or group of users. Some of chat applications provide users registration and profile manipulation.

3.6 File Transfer Protocol (FTP)

File transfer protocol, usually called FTP, is a utility for managing files across machines without having to establish a remote session with Telnet. FTP enables to transfer files back and forth, manage directions, and access electronic mail. FTP is not designed to enable access to another machine to execute programs, but it is the best utility for file transfer.

FTP uses two TCP channels. TCP port 20 is the data channel, and port 21 is the command channel. FTP is different from most other TCP/IP application programs in that it does uses two channels, enabling simultaneous transfer of FTP commands and data.

FTP transmission Modes

The transmission modes specify how FTP transmits the file across the TCP data connection. FTP defines three transmission modes;

NOTES

- Block mode
- Compressed mode and
- Stream mode

3.7 Telnet

The Telnet (Telecommunication network) program is intended to provide a remote login or virtual terminal capability across a network. In other words, a user on machine A should be able to log into machine B anywhere on the network, and as far as user is concerned, it appears that the user is seated in front of the machine B. The telnet service is provided through TCPs port number 3. The term Telnet is used to refer to both the program and the protocol that provide these services.

When two machines communicate using Telnet, Tenet itself can determine and set the communications and terminal parameters for the session during connection phase.

The Telnet protocol includes the capability not to support a service that one end of the connection cannot handle. When a connection has been established by Telnet, both ends have agreed upon a method for the two machines to exchange information, taking the load off the server CPU for a sizable amount of this work. The figure 3.2 shows the Microsoft telnet client window.

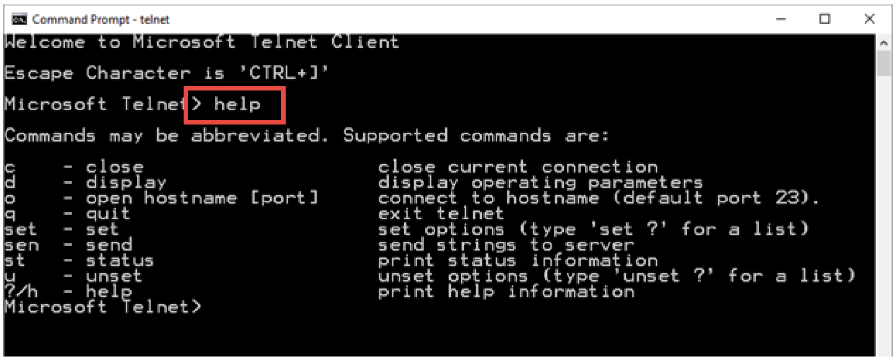


Figure 3.2 Microsoft Telnet Client Window

Telnet involves a process on the server that accepts incoming request for a Telnet Session. On UNIX systems, this process is called tented. On Windows NT and other PC based operating systems, a Telnet Server program is usually involved. The client runs a program, usually called telnet that attempts the connection to the server.

3.8 Check Your Progress Questions

1 What do you meant by File Transfer Protocol?
2. List various types of FTP transmission modes.
3. Write short note on: Telnet

3.9 Answers to Check Your Progress Questions

1. File transfer protocol, usually called FTP, is a utility for managing files across machines without having to establish a remote session with Telnet. FTP enables to transfer files back and forth, manage directions, and access electronic mail.

2. Block mode, Compressed mode and Stream mode

3. The telnet (Telecommunication network) program is intended to provide a remote login or virtual terminal capability across a network. In other words, a user on machine A should be able to log into machine B anywhere on the network, and as far as user is concerned, it appears that the user is seated in front of the machine B.

3.10 Summary

Usenet Newsgroup is storage with in the usenet system where the users from different geographical locations post their messages over internet. Internet Relay Chat (IRC) is an application layer protocol which is used to communicate in the form of text. File transfer protocol, usually called FTP, is a utility for managing files across machines without having to establish a remote session with Telnet.

3.11 Key Words

- **Chatting** is to talk in a way with one person with other or group of persons over the telecommunication especially internet.

- **The telnet** (Telecommunication network) program is intended to provide a remote login or virtual terminal capability across a network.

NOTES

3.12Self-Assessment Questions and Exercises

1. Name any two chat servers.
2. Write short note on Messaging and chatting
3. Briefly explain about Telnet communication

3.13 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998.

BLOCK 2
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

UNIT – 4
BASIC CONCEPTS OF OBJECT ORIENTED PROGRAMMING

Structure

4.1 Introduction

Computer programs consists of two elements namely code and data. A program can be organized around its code and data. There are two paradigms that reveal how a program is constructed. They are procedure oriented and Object oriented programming. Procedure oriented programming basically consists of writing a list of instructions for a computer to follow and organizing these instructions into functions. Little emphasis is given to data. More emphasis is on doing procedure. Larger programs are divided into functions. These functions share global data. Data move openly around the system from function to function. It uses top down approach in program design. The procedural languages BASIC,COBOL, FORTRAN and C adopt this approach. Object

NOTES

Oriented Programming is a new paradigm with various features such as structured programming, reliability and data security. There are many object oriented programming languages such as C++ and Java.

4.2 Objectives

After going through the unit you will be able to;

- Know the fundamentals of Object Oriented Programming (OOP)
- Understand the basic concepts of Object Oriented Programming
- Discuss about the applications of Java
- Know the advantages of Java Programming.

4.3Basic concepts / Principles of OOP

To manage the increasing complexity the object oriented programming was conceived. This approach organizes a program around its data called objects and a set of well defied interfaces to that data. OOP treats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the functions that operate on it and protects it from accidental modifications from outside function. OOP allows us to decompose a problem into a number of entities called objects and then builds data and functions around these entities. An OOP language such as C++ and Java supports the OOP approaches.

Java is an object oriented programming and to understand the functionality of OOP in Java, we first need to understand several fundamentals related to objects. These include class, object, inheritance, encapsulation, abstraction, polymorphism etc.

Data Abstraction

The process of abstraction in Java is used to hide certain details and only show the essential features of the object.

Class

A class is a collection of data and methods that operate on that data. A class is a template of an object. It is the central point of OOP and that contains data and codes with behavior. In Java everything happens

within class and it describes a set of objects with common behavior. The class definition describes all the properties, behavior, and identity of objects present within that class.

Object

An Object is an Instance of a Class. Objects are created by generating an instance of a class or in other words instantiating a class. There can be more than one object for a particular class. Objects are the basic unit of object orientation with behaviour, identity. Figure 1.1 shows the representation of class and objects.

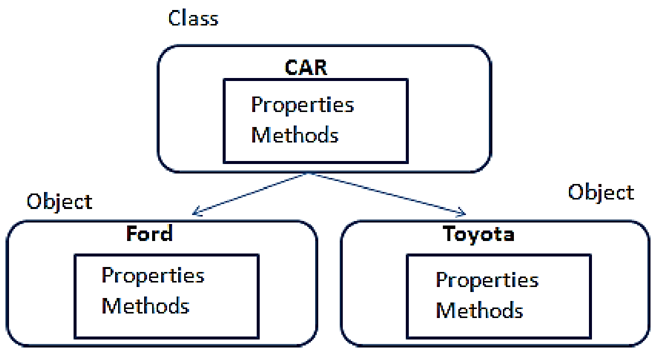


Figure 1.1 Class and its objects

Data Abstraction

Data abstraction defines the essential components (data members / attributes / properties such as size, price supplier and methods / membership function prototypes such as calculate()) excluding the implementation of it. It defines only what is to be implemented but excluding how. In object oriented programming class defines the abstraction of data. The data members are sometimes called as attributes or properties. Methods of a class are also called as membership function or data members.

Data Encapsulation

Combining data and its function into a single unit is called Data Encapsulation. Data encapsulation, sometimes referred to as data hiding, is the mechanism whereby the implementation details of a class are kept hidden from the user. This idea of hiding the details away from the user and providing a restricted, clearly defined interface is the underlying theme behind the concept of an abstract data type.

NOTES

Inheritance

Inheritance is the process by which child object can get the properties of parent object. In object-oriented programming, inheritance is a way to form new classes (instances of which are called objects) using classes that have already been defined. The new classes, known as derived classes, take over (or inherit) attribute and behavior of the pre-existing classes, which are referred to as base classes (or ancestor classes). It is intended to help reuse existing code with little or no modification.Figure 1.2 shows the process of inheritance.

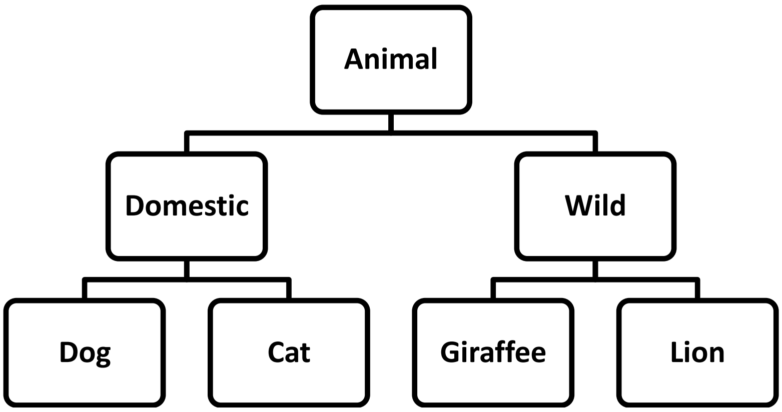


Figure 1.2 Inheritance

There are various types of inheritance. They are single inheritance, mulit-level inheritance, multiple inheritance and hybrid.

Polymorphism

The word polymorphism is Greek and literally means “many forms.” Polymorphism is a term that describes a situation where one name may refer to different methods. In java there are two type of polymorphism. They are overloading type (compile time polymorphism) and overriding type. (run time polymorphism). The figure 1.3 shows two types of polymorphism.

NOTES

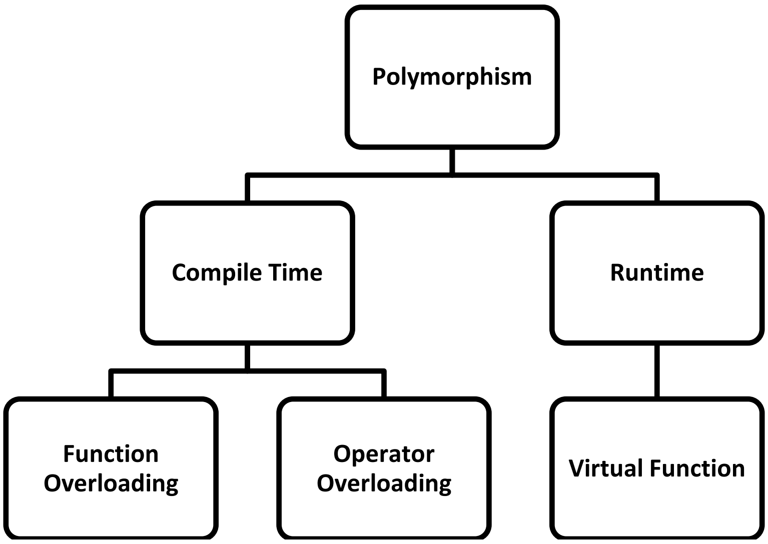


Figure 1.3 Two types of Polymorphism

Overloading occurs when several methods have same names with different method signature. Overloading is determined at the compile time.Overriding occurs when a class method has the same name and signature as a method in parent class.

A single function name area can be used to handle different number and different type of arguments like area(a) for square ,area(b,h) for triangle and area(rad) for circle. When you override methods, java determines the proper methods to call at the program's run time, not at the compile time. Figure 1.4 shows the function overloading of area() method.

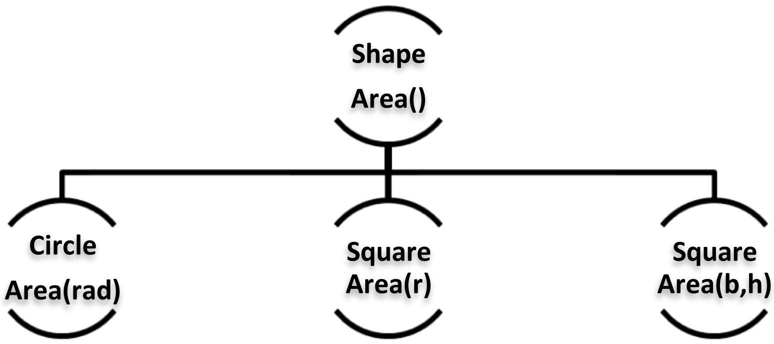


Figure 1.4 Function Overloading

NOTES

Message Passing

Message Passing is another object oriented programming concept in which one object can communicate with other objects. Objects can communicate each other through message passing.

For example, Consumer and Producer relationship can be established using message passing mechanism. In this example consumer and producer are considered as two objects.

4.4Benefits

The Object oriented programming paradigm came into existence to resolve the disadvantages of structured programming language paradigms. The benefits of object oriented Programming includes;

- **Reusability:**Reusability of code can be achieved through inheritance. By deriving a new class from the existing parent class reusability can be achieved.
- **Secure and Robust:** The principles of data hiding help to write a code to secure the data in various applications.
- **Message Passing:**These systems used to communicate between one object with other objects.
- **Flexibility:**Through polymorphism flexibility in code can be achieved using overloading and virtual function.
- **Modularity:** It is very easy partition the project into various sub modules.
- **Complexity:**The complexity of the software is reduced.
- OOP systems can be upgrade into small to large scale systems.
- It is possible to create multiple objects of the same class

4.5Applications

Object oriented programming has featured many applications from 1960's. Some of the applications of Object oriented Programming for software developments are;

- **Client Server systems :** Object Oriented Programming is used to develop client side (HTML, Javascript) and server side (Servlets) programming.
- **Real time system design:** It is also used in real time system design such as railway, on-line shopping (e-commerce) and auction.
- **Simulation and modeling system:** Through object oriented programming simulation and modeling can be applied to map real world problems. Simula-67 and Smalltalk are two object-oriented languages are designed for making simulations.
- **Databases :** Objected oriented databases are nowadays developed using object oriented languages
- **Scripting :** OOP has also been used for developing HTML,XHTML and XML documents for the Internet. Python, Ruby and Java are the scripting languages based on object-oriented principles which are used for scripting.
- **Graphics Applications :** The development of games and graphics software can be implemented using object oriented programming languages.

Some other areas of applications include neural network, Cloud/distributed systems, Embedded Systems, Libraries, Operating systems, parallel programming system, browser software, CAD/CAM systems and banking applications.

4.6 Check Your Progress Questions

1) Define Data Encapsulation.
2) What are the types of Polymorphism?
3) What are the concepts of OOPs?

NOTES

4.7 Answers to Check Your Progress Questions

1. Mixing of data and its function into a single unit is called Data Encapsulation.
2. Run-time polymorphism and Compile time polymorphism
3. Class,Objects, Data Abstraction, overloading, inheritance, data hiding, polymorphism and message passing.

4.8 Summary

Java is an object oriented programming. The basic concepts of object oriented Programming include class, objects, data abstraction, overloading, inheritance, data hiding, polymorphism and message passing. There are various benefits such as reusability, secure and robust. Object oriented programming is used to develop various applications such as real-time, embedded and cloud/distributed systems.

4.9 Key Words

- **Class** is a collection of data and methods that operate on that data. A class is a template of an object.
- **Object** is an instance of a class.

4.10 Self Assessment Questions and Exercises

1. Explain in detail about the fundamental concept of Object Oriented Programming.
2. Briefly explain about various applications of Java.
3. What are the benefits of Object Oriented Programming? Explain.

4.11 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.

4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998.

UNIT 5
JAVA EVOLUTION

Structure

5.1 Introduction

Java is a programming language derived from Oak language. It has many features such as robust, secure and platform independent. It differs from C and C++ in many ways. It also supports for internet programming. It has special environment to work with console and internet. This unit discuss about the evolution of java and its features.

5.2 Objectives

After going through the unit you will be able to;

- Understand Java Evolution
- List out the Features of Java
- Know How java differs from C and C++
- Learn about Java and internet
- Identify Java support system
- Discuss about Java environment

5.3Java Evolution

The Java programming Language evolved from a language named **Oak.** Oak was developed in the early nineties at *Sun Microsystems as a platform-independent language* aimed at allowing entertainment appliances such as video game consoles and VCRs to communicate. Oak was first slated to appear in television set-top boxes designed to provide video-on-demand services. As Oak's developers began to recognize this trend, their focus shifted to the Internet and WebRunner, an Oak-enabled browser, was born. Oak's name was changed to Java and WebRunner became the HotJava web browser.

5.4 Features of Java

Java has many features, where it follows object oriented programming and platform independence, the features are,

1. **Java is Interpreted: -** Strictly speaking java is interpreted although in reality java is both interpreted and compiled. A programmer first compiles java source code into byte code using **java compiler**. These byte codes are binary and architecturally neutral(platform independent)
2. **Java is platform independent and portable: -** Java programs can be executed easily from one type of computer to another.
3. **Java is Object Oriented: -** Java is a true object oriented language. Almost everything in java is an object. Java comes with an extensive set of classes arranged in packages that we can use in our programs by Inheritance.
4. **Java is robust and secure: -** It provides many safeguards to ensure reliable code. Java incorporates the concept of exception handling which captures serious errors and eliminate the errors.
5. **Java is distributed: -** Java is designed as a distributed language for creating applications on network. Java application can open and access remote objects on Internet as in a local system.
6. **Java is familiar, simple and small: -** Java does not use pointers, pre processors header files, goto statement etc. It also eliminates operator overloading and multiple inheritance.

NOTES

7. **Java offers high performance:-**Java architecture is designed to reduce overheads during run time. The multi threading enhances the execution speed of java program.
8. **Java is dynamic and extensible: -** Java is capable of dynamically linking in new class methods and objects. Java programs supports functions written in other languages such as C and C++. Native methods are linked dynamically at run time.

5.5 How java differs from C and C++

The major difference between C and C++ is that C is a procedural programming language and does not support classes and objects, while C++ is a combination of both procedural and object oriented programming language; therefore C++ can be called a hybrid language.

5.6 Java and internet

Java is strongly associated with the internet because of the first application program is written in Java was hot Java.Web browsers are used to run applets on the internet.

Internet users can use Java to create applet programs & run then locally using a Java-enabled browser such as hot Java.Java applets have made the internet a true extension of the storage system of the local computer.

- Java communicates with a web page through a special tag called .
- Java user sends a request for an HTML document to the remote computers net browser.
- The web-browser is a program that accepts a request, processes the request and sends the required documents.
- The HTML document is returned to that user browser.
- The document contains the applet tag which identifies the applet. The corresponding applet is transferred to the user computer.
- The Java enabled browser on the user's computer interprets the byte code and provide output.

5.7 Java support system

The term Java actual refers to more than just a particular language like C or Pascal. Java encompasses several parts, including the following

A high level language

The Java language is a high level one that at a glance looks very similar to C and C++ but offers many unique features of its own.

Java bytecode

A compiler, such s Sun's javac, transforms the Java language source code to bytecode that runs in the JVM.

Java Virtual Machine (JVM)

A program, such as Sun's java, that runs on a given platform and takes the byte code programs as input and interprets them just as if it were a physical processor executing machine code. JVM is the heart of java which is a virtual computer that resides in the memory only. The JVM enables java programs to be executed on various types of platforms. The JVM is the very reason that java is portable. JVM provides a layer of abstraction between the compiled java program and the underline hardware platform and the OS.

5.8 Java environment

A compiler converts the java program into an intermediate language representation called **BYTECODE** which is platform independent. A java file will have the extension .java. For example Hello.java. When this file is compiled we get a file called Hello.class.

This class file is run using an interpreter as and when necessary.

```
    class Hello1
{
          public static void main(String args[])
{
            system.out.println("Hello java world");
          }
}
```

The steps for compiling and running java program are as follows;

```
C:\java> javac Hello1.java
C:\java> java Hello1
Hello java world.
```

NOTES

The following figure 5.1 shows the architecture of java execution environment.

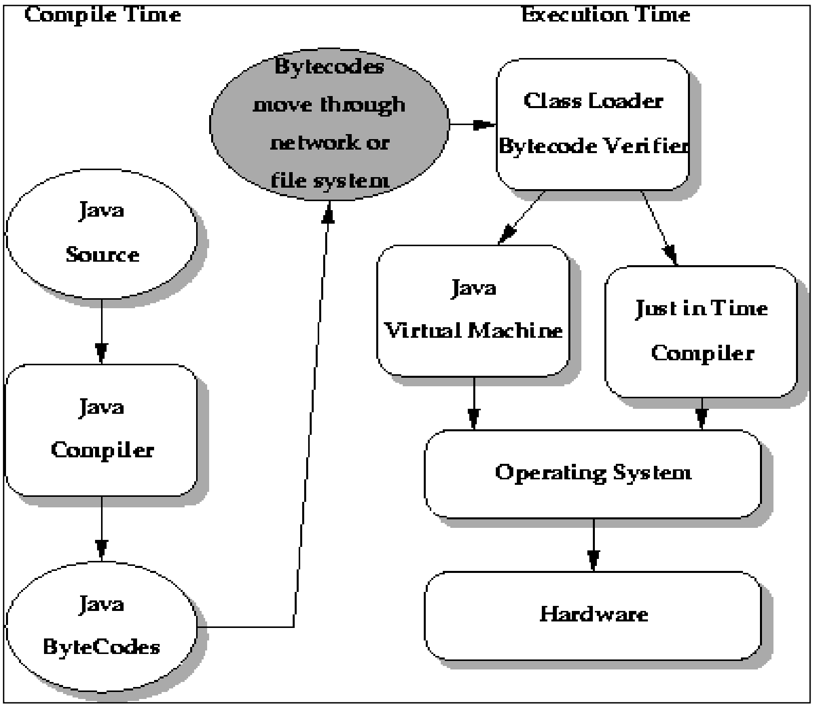


Figure 5.1 shows the java execution architecture

The concept of "**Write Once, run anywhere**" is possible in java. The java program can be compiled on any platform having a java compiler. The resulting **byte-codes** can be then run on any operating system on any machine like windows, Unix etc.. The above diagram illustrate how does it work.

Downloading and Installing Java on Windows:

1. Download the latest Version of Java SDK and install on your system.

2. Accept all of the defaults and suggestions, but make sure that the location where Java will be installed is at the top level of your C: drive. Click on "Finish." You should have a directory (folder) named C:\j2sdk1.5.0_04, with subfolders C:\j2sdk1.5.0_04\bin and C:\j2sdk1.5.0_04\lib

3. Modify your system variable called "**PATH**" (so that programs can find where Java is located).To does this for Windows 2000 or XP, either right-click on the My Computer icon or select "System" on the control panel. When the window titled "System Properties" appears, choose the

NOTES

tab at the top named "Advanced." Then, click on "Environment Variables." In the bottom window that shows system variables, select "Path" and then click on "Edit..." Add C:\j2sdk1.5.0_04\bin as the first item in the list. Note that all items are separated by a semicolon, with no spaces around the semicolon. You should end up with a path variable that looks something like C:\j2sdk1.5.0_04\bin;C:\WINNT\system32;C:\WINNT;C:\WINNT\system32\Wbem

For Windows 98 or ME, open the file AUTOEXEC.BAT in Notepad. You should find a line in this file that begins SET PATH=...Modify this line to add C:\j2sdk1.5.0_04\bin; immediately after the equals sign.

4. Modify or create a system variable called "**CLASSPATH,**" as follows. In the lower "System Variables" pane choose "New..." and type in Variable Name "CLASSPATH" and value (note that it begins with dot semicolon) .;C:\j2sdk1.5.0_04\lib

5. To test Java to see if everything is installed properly, open a command window (a DOS window) and type the command "javac" The result should be information about the Usage of javac and its options. If you get a result that "'javac' is not recognized as an internal or external command, operable program or batch file" then there is a problem and Java will not work correctly.

Note:

The name of the file and the name of the class must be same. The names are also case sensitive. It is a common practice to use a capital letter at the beginning of the name for a class.

5.9 Check Your Progress Questions

1. What do you meant by platform independent?
2. Is Java is capable of dynamically linking? Justify
3. What do you meant by byte code in java?

NOTES

5.10 Answers to Check Your Progress Questions

1. Java programs can be executed easily from one type of computer to another.
2. Java is capable of dynamically linking in new class methods and objects. Java programs supports functions written in other languages such as C and C++. Native methods are linked dynamically at run time.
3. A compiler converts the java program into an intermediate language representation called BYTECODE which is platform independent.

5.11 Summary

The Java programming Language evolved from a language named **Oak.**Java has many features, where it follows object oriented programming and platform independence

5.12 Key Words

- **Platform Independent -**Java programs can be executed easily from one type of computer to another.

- **Bytecode -** A compiler converts the java program into an intermediate language representation called bytecode which is platform independent.

5.13 Self-Assessment Questions and Exercises

1. Explain in detail about the features of java programming.
2. Elucidate about the java working environment.
3. Briefly explain about java support system.
4. How java differs from C++?

5.14 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004

NOTES

2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998.

NOTES

UNIT 6
OVERVIEW OF JAVA LANGUAGE

Structure

6.1 Introduction

The Java programming Language evolved from a language named Oak. Oak was developed in the early nineties at Sun Microsystems as a platform-independent language aimed at allowing entertainment appliances such as video game consoles and VCRs to communicate.

6.2 Objectives

After going through the unit you will be able to;

1. To understand the fundamentals of java programming.
2. To learn about Constants, variables and various data types.
3. To know about Operators and expressions in java
4. To discuss about various branching and Looping structure for decision making.

6.3 Overview of Java

Java was started by JAMES GOSLING team at SUN MICROSYSTEMS INC. in 1991. The main purpose of java is embedded systems like electronic devices like microwave oven, remote control system. Java was emerging to play a role in the administration of Internet.The popularity and the richness of the Java language is the set of packages that come bundled with Java Developer Kit (JDK).

Example:-

java.applet- These are the classes for developing applets
java.awt Abstract Windowing Tool Kit classes for GUI interface
java.net classes for networking, URL's client server sockets
java.io classes for various types of inputs & outputs

The term Java actual refers to more than just a particular language like C or Pascal. Java encompasses several parts, including the following

A high level language

The Java language is a high level one that at a glance looks very similar to C and C++ but offers many unique features of its own.

Java bytecode

A compiler, such s Sun's javac, transforms the Java language source code to bytecode that runs in the JVM.

Java Virtual Machine (JVM)

A program, such as Sun's java, that runs on a given platform and takes the byte code programs as input and interprets them just as if it were a physical processor executing machine code. JVM is the heart of java which is a virtual computer that resides in the memory only. The JVM enables java programs to be executed on various types of platforms. The JVM is the very reason that java is portable. JVM provides a layer of abstraction between the compiled java program and the underline hardware platform and the OS.

NOTES

6.4 Constants, Variables and data types

Constants

Constants are identifiers with values that can't be changed. The value is assigned to a constant when it is declared.

- Integer Constants - 0 to n
- Floating point Constants - 3.14
- Boolean Constants - 0, 1, true, false
- Character Constants - characters, alphanumeric, escape sequences
- String Constants - "HELLO WORLD"

Variables

Variables are places in the computer's memory where you store the data for a program. Each variable is given a unique name which you refer to it with.

Declaring variables

You can declare a variable either inside the class curly brackets or inside the main method's curly brackets. You declare a variable by first saying which type it must be and then saying what its name is. A variable name can only include any letter of the alphabet, numbers and underscores. Spaces are not allowed in variable names. You usually start a variable name with a lower case letter and every first letter of words that make up the name must be upper case. for Example,

```
public class VariablesExample{
  public static void main(String[] args)
  {
    int var1;
  }
}
```

If you want to declare more than one variable of the same type you must separate the names with a comma.

```
public class VariablesExample
{
  public static void main(String[] args)
  {
```

```
    int var1,var2;
  }
}
```

Scope of variables

Variables can be defined within any block. A block defines a scope. Each time you start a new block you are creating a new scope. A scope ultimately determines what objects visible to other parts of the program.

Three types of variables in java:-

1. Instance variables
2. class (or) static variables
3. local variables

1.Instance variables:-

Instance and class variables are declared inside a class. They are created when the objects are instantiated and therefore they are associated with close objects.

2. Class (or) static variables:-

Class variables are global to a class and belong to an entire set of objects that the class creates. Only one location is created for each class variable. If an object changes value of a static data member then the changed value is available for all other objects of that class.

3. Local variables:-

Variables declared and used inside methods are called local variables because they are not available for use outside the method definition Local variables can also be declared inside program blocks. When the program control leaves a block all the variables in the block will cease to exist.

Data Types

There are 8 *primitive data types*. The 8 primitive data types are numeric types. The names of the eight primitive data types are byte,short,int,long,float,double,char and boolean.

NOTES

Type	Values	Size	Default
Byte	signed integers	8 bits	0
Short	signed integers	16 bits	0
Int	signed integers	32 bits	0
Long	signed integers	64 bits	0l
Float	IEEE 754 floating point	32 bits	0.0f
Double	IEEE 754 floating point	64 bits	0.0d
Char	Unicode character	16 bits	\u0000
Boolean	true,false	1 bit used in 32 bit integer	false

Example

```
public class main1
{
   public static void main(String[] args)
{
      int days = 468;
       System.out.println("Days Passed: "+days);
   }
}
```

6.5 Operators and Expressions

There are eight types of operators in java. They are Assignment operators, Arithmetic operators, Increment / Decrement operators, Relational operators, Conditional operators, Bitwise operators ,Boolean operators, and Special operators.

Assignment Operators

The basic assignment operator = is used to assign a value to a variable or initialize a variable. The following assignment operations can be performed in java.

NOTES

Assignment Operators x operation= y is equivalent to x = x operation y x and y must be numeric or char types except for "=", which allows x and y also to be object references. In this case, x must be of the same type of class or interface as y. If mixed floating-point and integer types, the rules for mixed types in expressions apply.	
=	Assignment operator. x = y; y is evaluated and x set to this value. The value of x is then returned.
+=, -=, *=, /=, %=	Arithmetic operation and then assignment, e.g. x += y; is equivalent to x = x + y;
&=, |=, ^=	Bitwise operation and then assignment, e.g. x &= y; is equivalent to x = x & y;
<<=, >>=, >>>=	Shift operations and then assignment, e.g. x <<= n; is equivalent to x = x << n;

Arithmetic Operators

The following arithmetic operations can be performed in java.

NOTES

Arithmetic Operators x and y are numeric or char types. If mixed floating-point and integer types, then floating-point arithmetic used and a floating-point value returned. If mixed integer types, the wider type is returned. If double and float mixed, double is returned.	
x + y	Addition
x - y	Subtraction
x * y	Multiplication
x / y	Division.
x % y	Modulo - remainder of x/y returned.
-x	Unary minus Negation of x value

Example to add two numbers:

```
/* program to add two numbers */
class addnum{

public static void main(String args[]){
int a =10;
int b = 20;
int sum;
sum = a+b;
System.out.println("sum="sum);
}
}
```

Compiling and Executing the above program is :

```
C:\java>javac addnum.java
C:\java>java addnum
sum = 30
```

Increment & Decrement operators

These operators are placed either before the variable or after the variable name.

The example below shows the use of these operators.

rement & Decrement operators x and y are numeric (FP & integer) or char types.	
x++	Post-increment : add 1 to the value. The value is returned *before* the increment is made, e.g. x = 1; y = x++; Then y will hold 1 and x will hold 2
x--	Post-decrement : subtract 1 from the value. The value is returned *before* the decrement is made, e.g. : x = 1; y = x--; Then y will hold 1 and x will hold 0.
++x	Pre-increment : add 1 to the value. The value is returned *after* the increment is made, e.g. x = 1; y = ++x; Then y will hold 2 and x will hold 2.
--x	Pre-decrement : subtract 1 from the value. The value is returned *after* the decrement is made, e.g. x = 1; y = --x; Then y will hold 0 and x will hold 0.

```
/*Program using increment and decrement operators */
Class InDecOpr{
public static void main(String args[]){
        int x1=5;
        int x2=5;
        int y1,y2;
        y1=x++;
        System.out.println("x1= '+x1+" y1="+y1);
        y2=++x2;
        System.out.println("x2= '+x2+" y2="+y2);}
}

C:\java> javac IncDecopr.java
C:\java> java IncDecopr
        x1 = 6 y1= 5
        x2 =6 y2=6
```

NOTES

Relational Operators

The following Boolean operations can be performed in java.

Relational Operators x and y are numeric or char types only except for "==" and "!=" operators, which can also compare references. If mixed types, then the narrower type converted to wider type. Returned value is boolean true or false.	
x < y	Is x less than y ?
x <= y	Is x less than or equal to y ?
x > y	Is x greater than y ?
x >= y	Is x greater than or equal to y ?
x == y	Is x equal to y ?
x != y	Is x not equal to y ?

The example below prints larges of two numbers

```
/* program to find the largest of two numbers */
class largest
{
public static void main(String args[])
{
int x1=10;
int x2=20;

if(x1>x2)
        System.out.println("x1 is larger than x2");
Else    System.out.println("x2 is larger than x1");
}}
```

Conditional /Ternary Operator

The following Conditional operations can be performed in java.

Conditional /Ternary Operator		
x=*Boolean*?y:x	Conditional Operator	The first operand - *boolean* - is a boolean variable or expression. First this boolean operand is evaluated. If it is true then the second operator evaluated and x is set to that value. If the boolean operator is false, then the third operand is evaluated and x is set to that value.

Bitwise Operators

A bitwise operator allows you to perform bit manipulation on data. The following Bitwise operations can be performed in java.

Bitwise Operators x and y are integers. If mixed integer types, such as int and long, the result will be of the wider type		
~x	Compliment	Flip each bit, ones to zeros, zeros to ones
X & y	AND	AND each bit a with corresponding bit in b
X | y	OR	OR each bit in a with corresponding bit in b
X ^ y	XOR	XOR each bit in x with corresponding bit in y
X << y	Shift left	Shift x to the left by y bits. High order bits lost.Zero bits fill in right bits.
X >> y	Shift Right - Signed	Shift x to the right by y bits. Low order bits lost. Same bit value as sign (0 for positive numbers, 1 for negative) fills in the left bits.
X >>> y	Shift Right - Unsigned	Shift x to the right by y bits. Low order bits lost. Zeros fill in left bits regardless of sign.

The following example shows subtraction using one's complement operation.

```
class sub
{
public static void main(String args[])
{
        int x1=25;
        int x2=20;
int temp,result;

//To find one's complement
temp =~x2;

//To find two's complement
```

```
temp=temp+1;
result = x1+temp;
System.out.println(“x1-x2=”+result);
Sytem.out.println(“x1=x2=”+result);
}
}
```

Boolean Operators

The following Boolean operations can be performed in java.

Boolean Operators x and y are boolean types. x and y can be expressions that result in a boolean value . Result is a boolean true or false value.		
X && y	(short circuit) Conditional AND	If both x and y are true, result is true. If either x or y are false, the result is false If x is false, y is not evaluated.
X & y	Boolean AND	If both x and y are true, the result is true. If either x or y are false, the result is false Both x and y are evaluated before the test.
X || y	(short circuit) Conditional OR	If either x or y are true, the result is true. If x is true, y is not evaluated.
X | y	Boolean OR	If either x or y are true, the result is true. Both x & y are evaluated before the test.
!x	Boolean NOT	If x is true, the result is false. If x is false, the result is true.
X ^ y	Boolean XOR	If x is true and y is false, the result is true. If x is false and y is true, the result is true. Otherwise, the result is false. Both x and y are evaluated before the test.

Class and Object Operators

The following class and object operations can be performed in java.

Class and Object (Special) Operators		
x instanceof *c*	Class Test Operator	The first operand must be an object reference. *c* is the name of a class or interface. If x is an instance of type c or a subclass of c, then true returned. If x is an instance of interface type c or a sub-interface, then true is returned. Otherwise, false is returned.
new c(args)	Class Instantiation	Create an instance of class c using constructor c(args)
"."	Class Member Access	Access a method or field of a class or object : o.f - field access for object o o.m() - method access for object o

Operator Precedence

When more than one operator is used in an expression, java has established operator precedence to determine the order in which the operators are evaluated. For example, consider the following example, Cal = 10 + 3 * 2 – 12 / 4.In this expression multiplication and division operation have higher priority than addition and multiplication. Hence they are performed first. After that the right hand side becomes 10 + 6 – 3

NOTES

<table>
<tr><td colspan="2">Operator Associatively
The following operators have Right to Left associatively. All other operators (see precedence table above) are evaluated left to right.</td></tr>
<tr><td>=
*=
/=
%=
+=
-=
<<=
>>=
>>>=
&=
^=
|=</td><td>?:
new
(type cast)
++x
--x
+x
-x
~
!</td></tr>
</table>

<table>
<tr><td colspan="15">Operator Precedence

The larger the number, the higher the precedence.</td></tr>
<tr><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
<tr><td>=
*=
/=
%=
+=
-=
<<=
>>=
>>>=
&=
^=
|=</td><td>?:</td><td>||</td><td>&&</td><td>|</td><td>^</td><td>&</td><td>==
!=</td><td><
<=
>
>=</td><td><<
>>
>>></td><td>+
-</td><td>*
/
%</td><td>new
(type)</td><td>++x
--x
+x
-x
~
!</td><td>.
[]
(args)
x++
x--</td></tr>
</table>

6.6 Decision Making and Branching

Decision making and branching are needed for the user to execute a set of statements repeatedly for a specific number of times or till a particular condition is satisfied. Also the sequence of execution need to be altered depending of the logic of the Program.

For example, it may be required to find the area of the triangle for many values of the sides. To perform these tasks, the control statements come in handy.

Conditional Control (Branching) Statements

The conditional control statement can be broadly classified as follows:

i) Conditional Execution
ii) Selection
iii) Transfer statements
iv) Looping

i) Conditional Execution (if – else statement)

The 'if-else' statement is used for conditional statement. This statement executes on a logical test and performs one of the two possible actions depending on the result of the test. If the result is true then the statements in the 'if' construct are executed, other wise the statements in the 'else' construct are executed.
The general form of 'if-else' is given below:
General format 1 :

```
if (condition) {
statements;
}
```

General format 2 :

```
if (condition) {
statements;
}
else {
statements;
}
```

NOTES

Example:

```
class IfElseDemo {
    public static void main(String[] args) {
        int testscore = 76;
        char grade;
        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else if (testscore >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

The output from the program is: .

```
Grade = C
```

You may have noticed that the value of testscore can satisfy more than one expression in the compound statement: 76 >= 70 and 76 >= 60. However, once a condition is satisfied, the appropriate statements are executed (grade = 'C';) and the remaining conditions are not evaluated.

Nested if statement

The conditional if-else statement can be used one within the other and these statements are said to be nested. The general form is as follows :

```
if (condition 1){
        if(condition 2)
          {
          Statements
        }
        else
          {
          Statements
        }
}
```

```
else{
        if(condition 3)
              {
                Statements
              }
              else{
                Statements
                }
      }
```

Example

```
if (length < 10)
    {
System.out.println("Small");
     }
else {
     if (length < 20)
           {
System.out.println("Medium");
     }
else
    {
System.out.println("Large");
    }
}
```

Selection (Switch statement)

Selection statement allows the user to select a particular group of statements from several groups. The switch statement is used for this purpose.

The switch statement is an extension of if-else statement. In this case of an if-else statement, the maximum number of branches is restricted to 2, whereas with the help of switch statement you can have more number of branches,

Syntax :

```
switch (expression)
{
case 1:
       code block1
   case 2:
       code block2
```

NOTES

```
        .
        .
        .
        default:
            code default;
    }
```

The expression to the switch must be of a type byte, short, char, or int. Then there is a code block following the switch statement that comprises of multiple case statements and an optional default statement.

The execution of the switch statement takes place by comparing the value of the expression with each of the constants. The comparison of the values of the expression with each of the constants occurs after the case statements. Otherwise, the statements after the **default** statement will be executed.

Now, to terminate a statement following a switch statement uses break statement within the code block. However, it is an optional statement. The break statement is used to make the computer jump to the end of the switch statement. Remember, if we won't use break statement the computer will go ahead to execute the statements associated with the next case after executing the first statement.

For example

```
/*Displays the name of the day, based on the value of week
class switchex
{
  public static void main(String[] args)
{
    int week = 5;
       switch (week){
          case 1:  System.out.println("monday"); break;
          case 2:  System.out.println("tuesday"); break;
          case 3:  System.out.println("wednesday"); break;
          case 4:  System.out.println("thursday"); break;
          case 5:  System.out.println("friday"); break;
          case 6:  System.out.println("saturday"); break;
          case 7:  System.out.println("sunday"); break;

          default: System.out.println("Invalid week");break;
       }   }}
```

Output :

```
C:\javac> javac switch.java
C:\javac>java switch Friday
```

In this case, “friday” is printed to standard output.

Transfer Statements

Sometimes we use Jumping Statements in Java. Using for, while and do-while loops is not always the right idea to use because they are cumbersome to read. Using jumping statements like break and continue it is easier to jump out of loops to control other areas of program flow.

***break* statement**

break statement is used to terminate the loop once the condition is satisfied..

Syntax**:** break;

Example**:**

```
class BreakDemo
{
  public static void main(String[] args)
{
    for (int i = 0; i < 5; i++) {
      System.out.println(i);
      if (i==3) {
        break ;
}
  }
  }
}
```

Output :

```
BreakDemo
0
1
2
3
```

continue statement

Continue statement is just similar to the break statement in the way that a break statement is used to pass program control immediately after the end of a loop and the continue statement is used to force program control back to the top of a loop. The continue statement skips the current iteration of a for, while, or do-while loop. Lets see

NOTES

the same example of break statement but here we will use continue instead of break.

Syntax **:** continue;

Example**:**

```
class continueState
{
public static void main(String[] args)
{
    for (int i = 0; i < 5; i++) {
    System.out.println(i);
     if (i==3) {
    continue ;    } }} }
```

Output :

```
0
1
2
3
4
```

return statements:

It is a special branching statement that transfers the control to the caller of the method. This statement is used to return a value to the caller method and terminates execution of method. This has two forms: one that returns a value and the other that cannot return. The returned value type must match the return type of method.

Syntax**:**

```
return;

return values;
```

Example**:**

```
public static void  demo()
{
    System.out.println("welcome"+welcome());
}
static String welcome()
{
    return "java world";
}
```

Output : welcome java world

6.7 Looping Statements

There may be situations in a problem where a single statement or group of statements may have to be executed repeatedly, until a particular condition is satisfied. The following various looping statements are available in Java.

i) while statement
ii) do – while statement
iii) for statement

i) while statement

The 'while' statement continually executes a block of statements when a particular condition is true. Its syntax can be expressed as:

```
while (conditional expression) {
    statement(s)
}
```

The while statement evaluates *expression*, which must return a boolean value. If the expression evaluates to true, the while statement executes the *statement*(s) in the while block. The while statement continues testing the expression and executing its block until the expression evaluates to false.

Example **:** Using the while statement to print the values from 1 through 5 can be accomplished as in the following program:

```
/* To illustrate while statement
class WhileDemocode
 {
    public static void main(String[] args)
{
        int count = 1;
        while (count < 6) {
            System.out.println("Count is: " + count);
            count++;
        }
    }
}
```

NOTES

The **output** of this program is:

```
Count is : 1
Count is : 2
Count is : 3
Count is : 4
Count is : 5
```

ii) do - while statement

The Java programming language also provides a do-while statement, which can be expressed as follows:

```
do {
    statement(s)
} while (conditional expression);
```

The difference between do-while and while is that do-while evaluates its expression at the bottom of the loop instead of the top. Therefore, the statements within the do block are always executed at least once, as shown in the following Program:

Example

```
class DoWhileDemo
 {
    public static void main(String[] args)
{
        int count = 1;
        do {
            System.out.println("Count is: " + count);
            count++;
        } while (count <= 6);
    }
}
```

The **output** of this program is:

```
Count is : 1
Count is : 2
Count is : 3
Count is : 4
Count is : 5
```

iii) for statement

The for loop is the most versatile loop provided in java language. It accepts an initial condition and continues to loop until the condition is specified in the loop is met. In for loop, the initial value, increment value and the condition are specified in single statement. This reduces ambiguity and increases the readability of the program.

The general form of the for statement can be expressed as follows:

```
for (initialization; termination; increment)
{
statement(s)
}
```

When using this version of the for statement, keep in mind that:

- The *initialization* expression initializes the loop; it's executed once, as the loop begins.
- When the *termination* expression evaluates to false, the loop terminates.
- The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value.
-

The following program is an example to print the numbers 1 through 5 to standard output:

```
class ForDemo
{
    public static void main(String[] args)
{
        for(int i=1; i<6; i++)
{
            System.out.println("Count is: " + i);
        }
    }}
```

The **output** of this program is:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
```

NOTES

Nested Loops

The placing of one loop inside the body of another loop is called nesting. When you "**nest**" two loops, the outer loop takes control of the number of complete repetitions of the inner loop. While all types of loops may be nested, the most commonly nested loops are for loops.The following are the rules to be followed while implementing the nested for – loops

i) Each for the loops should have a unique index variable and should not coincide with the other index variables
ii) The loops should not overlap each other.
iii) The loops should be completely embedded with each other.

For example :

```
System.out.println("num2 "+"  "+ " num1");
for(num2 = 0; num2 <= 3;  num2++)
{
    for(num1 = 0; num1 <= 2; num1++)
    {
        System.out.println(num2 + "   " + num1);
    }
}
```

Output

```
num2        num1
0           0
0           1
0           2
1           0
1           1
1           2
2           0
2            1
2           2
3           0
3           1
3           2
```

6.8 Check Your Progress Questions

1. What do you meant by scope of variables?
2. What is the purpose of continue statement?

6.9 Answers to Check Your Progress Questions

1) Variables can be defined within any block. A block defines a scope. Each time you start a new block you are creating a new scope. A scope ultimately determines what objects visible to other parts of the program.
2) Continue statement is just similar to the break statement in the way that a break statement is used to pass program control immediately after the end of a loop and the continue statement is used to force program control back to the top of a loop.

6.10 Summary

Java was started by JAMES GOSLING team at SUN MICROSYSTEMS INC. in 1991. The main purpose of java is embedded systems like electronic devices like microwave oven, remote control system. Java was emerging to play a role in the administration of Internet.

There are eight types of operators in java. They are Assignment operators, Arithmetic operators, Increment / Decrement operators, Relational operators, Conditional operators, Bitwise operators ,Boolean operators, and Special operators.

Decision making and branching are needed for the user to execute a set of statements repeatedly for a specific number of times or till a particular condition is satisfied.

6.11 Key Words

- **Constants** are identifiers with values that can't be changed. The value is assigned to a constant when it is declared.

NOTES

- **Variables** are places in the computer's memory where you store the data for a program. Each variable is given a unique name which you refer to it with.

6.12 Self-Assessment Questions and Exercises

1. Differentiate between continue and break statements
2. Differentiate between while and for looping statements
3. Briefly explain about nested loops and nested if structure.

6.13 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth dition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998

BLOCK 3
CLASSES, OBJECTS AND METHODS

UNIT 7CLASS

Structure

7.1 Introduction

In object oriented programming class, object and methods play vital role. Class defines the structure of an object. Class consist of data member are membership functions. You can have any number of objects for a particular class. Class members can be accessed use class member access operator.

7.2 Objectives

After going through the unit you will be able to;

- To understand about class, objects, fields and methods
- To know about the inheritance and its types.

NOTES

- To discuss about constructors
- To learn about method overloading and method overriding
- To understand the nesting of methods and final

7.3Defining a class, fields and methods

A class is a collection of data and methods that operate on that data. A class is a template of an object. Class is a keyword.

Defining Class and fields

It has the following parts.

- Class Definition
- Creating instance and class variables
- Method definitions

General Format :

```
class classname
{

        access datatype instance variable1;
        access  datatype instance variable2;
                …..
                ……
        access datatype instance variableN;

        access returntype methodname1(arguments list);
        access returntype methodname2(arguments list2);
                …..
                …..
        access returntype methodnameN(arguments listN);
   }
```

Where access is access specified like public, private, default and protected

Example :

```
public class Circle
{
   public double x, y;   // The coordinates of the center
   public double r;      // The radius

   // Methods that return the circumference and area of the circle
   public double circumference()
```

```
   { return 2 * 3.14159 * r; }
   public double area()
   { return 3.14159 * r*r; }
}
```

Defining Methods

Methods can be defined inside a class. The methods contains the following parts,

- return type
- Name of the method
- A list of arguments
- Body of the method

General format:

```
access returntype methodname1(arguments list);
```

Where access is access specifier like public,private,default and protected.

Example

```
  void display()
{
      System.out.println("message");
    }
```

7.4Creating objects and accessing class members

An Object is an Instance of a Class. Objects are created by generating an instance of a class or in other words instantiating a class. There can be more than one object for a particular class.

Example :

```
Circle c;
c = new Circle();

   or

Circle c = new Circle();
Circle c1 = new Circle();
```

NOTES

Now you have created an instance of our Circle class In this new is the java Operator that creates the object.

Accessing Object Data

Now that we've created an object, you can use its data fields.
Circle c = new Circle();
c.x = 2.0; // Initialize our circle to have center (2, 2) and radius 1.0.
c.y = 2.0;
c.r = 1.0;
Using Object Methods
This is where things get interesting! To access the methods of an object, you use the same syntax as accessing the data of an object:
Circle c = new Circle();
double a;
c.r = 2.5;
a = c.area();
Take a look at that last line. We did not say:
    a = area(c);
We said:
    a = c.area();

7.5Constructors

Constructor is a special function used to initialize the objects.Every class has at least one its own constructor. Constructor creates an instance for the class. Constructor initiates (initialize) something related to the class's methods.

- Constructor is the method which name is same to the class.
- Constructors are invoked automatically when an object is created.
- Constructor methods do not have any return type.
- Typically they are used to set initial values to instance variables.

Example

```
class another
{
  int x,y;
  another(){   }
  int area()
{
    int ar = x*y;
    return(ar);  }
}
public class Construct{
```

NOTES

```
    public static void main(String[] args){
      another b = new another();
      b.x = 2;
 b.y = 3;
      System.out.println("Area of rectangle : " + b.area());
      System.out.println("Value of y in another class : " + b.y);
    }
}
```

Parameterized Constructors:

Similar to the methods Constructors also can have parameters. for example

```
class another{
  int x,y;
  another(int a, int b)
{
    x = a;     y = b; }
int area(){
    int ar = x*y;     return(ar);   }
}
public class Construct
{
   public static void main(String[] args)  {
     another a = new another(1,1);
     System.out.println("Area of rectangle : " + a.area());
     System.out.println("Value of x in another class : " + a.x);
}}
```

Constructor Overloading

You can have more then one constructor for a particular class. But the number or type of arguments has to be different.In this example you will see that how to implement the constructor feature in a class. This program is using two classes. First class is another and second is the main class which name is Construct. In the Construct class two objects (a and b) are created by using the overloaded another Constructor by passing different arguments and calculated. The area of the different rectangles is calculated by passing different values to the constructor.

Example

```
class another
```

NOTES

```
{
 int x,y;
 another(int a, int b)
{
  x = a;
  y = b;
 }
 another()
 {
 }
 int area()
{
  int ar = x*y;
  return(ar);
 }
}
public class Construct
{
  public static void main(String[] args)
  {
   another b = new another();
   b.x = 2;
   b.y = 3;
   System.out.println("Area of rectangle : " + b.area());
   System.out.println("Value of y in another class : " + b.y);
   another a = new another(1,1);
   System.out.println("Area of rectangle : " + a.area());
   System.out.println("Value of x in another class : " + a.x);
  }
}
```

A Constructor which doesn’t take any arguments is called as default constructor. Default constructors are automatically provided by java.

7.6Methods overloading

You can have same name for more than one method. The number of arguments and / or they type of arguments are to be different for creating two or more methods with the same name. The return types of the methods can be different as long as the parameter is different

NOTES

The following example below shows overloading of a method call as add. The first add() takes integers as arguments and returns the sum of a floating point number. The second add() takes floating point numbers as parameters and returns the sum as a floating point number. The third add() method takes string as arguments, and converts them to integer and then returns sum as flowing point number.

```
// Program to illustrate Overloading

class overloadingmethod
{
        static float add(int x , int y)
        {
          return x+y;
        }
        static float add(float t1,float t2)
        {
          return t1+t2;
         }
         static float add(String s1,String s2)
        {
            float sum;
             sum=Integer.parseInt(s1)+Integer.parseInt(s2);
             return sum;
         }
      public static void main(String args[])
    {
      int x = 100; int y=200;
      float m =15.5f;
      float n = 10.5f;
      String s1="125";
      String s2="145";
      System.out.println(add(x,y));
      System.out.println(add(m,n));
      System.out.println(add(s1,s2));
     }
  }
```

Output:

300.0
26.0
270.0

NOTES

7.7Static Methods and Members

A static method can be accessed without creating an instance of the class. If you try to use a non-static method and variable defined in this class then the compiler will say that non-static variable or method cannot be referenced from a static context.

Note :

Static method can call only other static methods and static variables defined in the class.

The this keyword can't be used in a static methods.

Example

```
 public class staticmethodex
{
 int i;
 static int j;
 public static void staticMethod()
{
   System.out.println("you can access a static method this way");
}
 public void nonStaticMethod()
{
   i=100;
   j=1000;
   System.out.println("non static method—cannot be called");
}
 public static void main(String[] args)
{
   //i=10;
    j=100;
    //non static methods cannot be called without instance
   //nonStaticMethod();
   staticMethod();// called without instance
  }
}
```

7.8Nesting of methods

A method which embeds another method is called nesting of method. For example.

```
void outer_method1()
{
int a,b;

void innermethod_display()
{
System.out.println("nested method")
}
}
```

7.9Inheritance

Inheritance is the mechanism through which you can derive classes/subclasses from other classes. This means that an object of a subclass can be used wherever an object of the superclass can be used.

The derived class is called as child class or the subclass or you can say the extended class and the class from which you are deriving the subclass are called the base class or the parent class. To derive a class in java the keyword extends is used.. The subclass inherits members of the superclass and hence promotes code reuse. The subclass itself can add its own new behavior and properties.

The **java.lang.Object** class is always at the top of any Class inheritance hierarchy.

The following kinds of inheritance are discussed here.

- SimpleInheritance
- MultilevelInheritance

Simple/Single Inheritance

Deriving a new class from existing one parent class is called single inheritance.The following example illustrates the **simple /single inheritance.**

Example**:**

NOTES

```
class superclass{
        void addnum(int x, int y)
        {
        int sum;
         Sum = x+y;
           System.out.pritnln("sum of numbers ="+sum);
        }
        void display(){
            System.out.pritnln("this is super class display ");
         }
   }

   class subclass extends superclass
   {
        public static void main(String args[])
        {
           subclass s1 = new subclass();
             s1.display();
             s1.adnum(10,20);
    }
}
```

Multilevel Inheritance

It is the enhancement of the concept of inheritance. When a subclass is derived from a derived class then this mechanism is known as the multilevel inheritance. The derived class is called the subclass or child class for its parent class and this parent class works as the child class for it's just above (parent) class. Multilevel inheritance can go up to any number of levels. for Example,

```
class A {
  int x;
  int y;
  int get(int p, int q){
     x=p;
y=q;
     return(0);
    }
    void Show()
{
      System.out.println(x);
      }
}

class B extends A
```

NOTES

```
{                   // level one
  void Showb(){
    System.out.println("B");
    }
}

class C extends B
{                   // level two
  void display(){
    System.out.println("C");
  }

  public static void main(String args[])
{
    A a = new A();
    a.get(5,6);
    a.Show();
    }
}
```

Multiple Inheritances

The mechanism of inheriting the features of more than one base class into a single subclass is known as multiple inheritances. Java does not support multiple inheritances directly but the multiple inheritances can be achieved by using the interface.

In Java Multiple Inheritance can be achieved through use of Interfaces by implementing more than one interface in a class.

7.10Method Overriding

When a subclass contains a method with the same name and signature as in the super class then it is called as method overriding.

Example

```
class superclass{
        void addnum(int x, int y){
        int sum;
         sum = x+y;
           System.out.println("sum of numbers ="+sum);
        }
void display(){
            System.out.println("super class  method is hidden by
subclass");
```

NOTES

```
        }
    }

    class subclass extends superclass{
        void display(){
            //hides super class display method
            System.out.println("sub class method is called");
        }
        public static void main(String args[]){
            subclass s1 = new subclass();
            s1.display();
            s1.addnum(10,20);
        }
    }
```

In this example display() method is defined in both superclass and subclass with same name and type signatures. When display() method is called with subclass object, it hides superclass method and calls subclass method.

Dynamic Method Dispatch

It is a mechanism by which when an overridden method is called by a super class object. Java determines the version of the method to be called based upon the object being used to at the time of call.

Example **:**

```
class superclass{
  public void fun1(int x){
    System.out.println("int in superclass");
  }
  public void fun1(int x, int y){
    System.out.println("int and int");
  }
}

class subclass extends superclass{
  public void fun1(int x){
    System.out.println("int in subclass");
  }
}
public class D{
  public static void main(String[] args){
    superclass obj;
    obj= new superclass(); // line 1
    obj.fun1(2);  // line 2 (prints "int in superclass")
    obj=new subclass();  // line 3
```

NOTES

```
obj.fun1(2);  // line 4 (prints "int in subclass")
  }
}
```

7.11final Variables,classes and methods

- A final variable cannot change its value (to define identifier as constant)
- A final class cannot be subclassed (to prevent inheritance)
- A final method cannot be overridden by any subclasses (to prevent overriding)

Variables are declared as final when their value does not change.

```
public final PI=3.14;
```

A final class cannot be sub classed by another class. For example java.lang.String is final. All methods in a final class are automatically final.

```
public final class A {}
```

then that means that A cannot be further extended or subclassed.

```
class B extends A {}  // will not work
```

A final method cannot be overridden by subclasses. This is used to prevent unexpected behavior from a subclass altering a method that may be crucial to the function or consistency of the class.

```
publicclass A {
publicfinal void myFinalMethod() {  }
 }
 Class B extends A{
void myFinalMethod() {  } // will not work

}
```

Abstract Classes

Sometimes you will want to create a super class that only defines the generalized form that will be shared by all of its sub classes leaving it to each sub class to fill in the details. **Java Abstract classes** are used to declare common characteristics of subclasses. Abstract classes are used to provide a template or design for concrete subclasses down the inheritance

NOTES

tree. Such objects would be used as abstract because an abstract class is not fully defined. Abstract classes are declared with the abstract keyword.

- An abstract class cannot be instantiated with **new** operator.
- It can only be used as a superclass for other classes that extend the abstract class.
- You cannot declare abstract constructors.
- You cannot declare abstract static members.

Any sub class of an abstract class must either implement all of the abstract methods in the super class, or be itself declared as abstract.For Example,

```
abstract class abstractclassdemo
{
        public void printHello()
{
       // non abstract method i.e. concrete method still allowed
       System.out.pritnln("Printing from abstract class concrete method");
   }
abstract void printme():
                  }
class testabstract extends abstractclassdemo{
                      public void printme(){
                      System.out.println("Implementation of abstract method");
                             }
                            public static void main (String args[]){
                              testabstract p = new testabstract();
                               p.printhello();
                               p.printme();
                     }
                   }
```

7.12 Check Your Progress Questions

1 What is the purpose of abstract classes?
2 What do you meant by dynamic method dispatch?

7.13 Answers to Check Your Progress Questions

1 Abstract class is used to declare common characteristics of subclasses. Abstract classes are used to provide a template or design for concrete subclasses down the inheritance tree.

2 It is a mechanism by which when an overridden method is called by a super class object. Java determines the version of the method to be called based upon the object being used to at the time of call.

7.14 Summary

Inheritance is the mechanism through which you can derive classes/subclasses from other classes. This means that an object of a subclass can be used wherever an object of the superclass can be used.

7.15 Key Words

class is a collection of data and methods that operate on that data.

- **Object** is an Instance of a Class.
- **Constructor** is a special function used to initialize the objects. Every class has at least one its own constructor. Constructor creates an instance for the class. Constructor initiates (initialize) something related to the class's methods.

7.16 Self-Assessment Questions and Exercises

1. Explain in detail about various types of inheritance with suitable example.
2. Briefly explain about constructor overloading.
3. What are abstract classes? Give example
4. Illustrate the use of final variables and methods.

7.17 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O’Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.

NOTES

7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998.

UNIT 8
ARRAYS, STRINGS AND VECTORS

Structure

8.1 Introduction

Variables are used to store one value at a time. When you want to store multiple values of same type in a single identifier array is used. Strings are special form of arrays which contains sequence of characters.

Vectors differ considerably from arrays. Arrays are of only one type of values and the number of elements cannot be changed. Vectors instead can hold a mix of class objects (they are, of course, all subclasses of Object.)

8.2 Objectives

After going through the unit you will be able to;

- Know about fundamentals of arrays, strings and vectors
- Understand about creating and using one dimensional and multidimensional arrays.
- Perform various string manipulation functions
- Know about various wrapper classes

- Discuss about creating and using vectors
- Learn about multiple inheritances

8.1 One dimensional Arrays

The java array enables the user to store the values of the same type in contiguous memory allocations. Arrays are always a fixed length abstracted data structure which cannot be altered when required. Arrays can hold data of similar type.

To use an array we require the following to be done.

- Declare a variable to be used as the array name
- Create an array object and assign to the variable

8.2 Creating One Dimensional Array

Array variable has a type and a valid Java identifier i.e. the array's type and the array's name. By type we mean the type of elements contained in an array. To represent the variable as an Array, we use [] notation. These two brackets are used to hold the array of a variable.for Example,

Examples

```
int[] array_name;     //declares an array of integers
String[] names;
int[][] matrix;    //this is an array of arrays
```

It is essential to assign memory to an array when we declare it. Memory is assigned to set the size of the declared array. for example:

```
int[] array_name = new int[5];
```

Here is an example that creates an array that has 15 elements.

```
public class Array{
public static void main(String[] args){
   float[] a = new float[15];
 }
}
```

Array Initialization

The "new" operator is used for the allocation of memory to the array object. The correct way to use the "new" operator is

```
String names[];
names = new String[10];
```

Here, the new operator is followed by the type of variable and the number of elements to be allocated. In this example [] operator has been used to place the number of elements to be allocated. For Example

```
public class Sum{
  public static void main(String[] args){
    int[] x = new int [101];
    for (int i = 0; i<x.length; i++ )   x[i] = i;
    int sum = 0;
    for(int i = 0; i<x.length; i++)
      sum += x[i];
      System.out.println(sum);
  }
}
```

In this example, a variable 'x' is declared which has a type array of int, that is, int[]. The variable x is initialized to reference a newly created array object. The expression 'int[] = new int[50]' specifies that the array should have 50 components. To know the length of the Array, we use field length, as shown.

Output for the given program:

```
C:\java>java Sum

5050
```

Instead of assigning memory to the array you can assign values to it instead. This is called initializing the array because it is giving the array initial values.

```
public class Array{
      public static void main(String[] args) {
        int[] a = {12, 23, 34, 45, 56};
      }
}
```

Using an array

You can access the values in an array using the number of the element you want to access between square brackets after the array's name. There is one important thing you must remember about arrays which is they always start at 0 and not 1. Here is an example of how to set the values for an array of 5 elements.

```
public class Array{
  public static void main(String[] args)   {
    int[] a = new int[5];
    a[0] = 10;
    a[1] = 20;
    a[2] = 30;
    a[3] = 40;
    a[4] = 50;   }}
```

A much more useful way of using an array is in a loop. Here is an example of how to use a loop to set all the values of an array to 0 which you will see is much easier than setting all the values to 0 separately.

```
public class Array{
  public static void main(String[] args) {
    int[] a = new int[5];
    for (int i = 0; i < 5; i++)          a[i] = 0;
  }
}
```

Sorting an array

Sometimes you will want to sort the elements of an array so that they go from the lowest value to the highest value or the other way around. Here is an example.

```
public class Array{
  public static void main(String[] args)   {
    int[] a = {3, 5, 1, 2, 4};
    int i, j, temp;
    for (i = 4; i >= 0; i--)
      for (j = 0; j < i; j++)
        if (a[j] > a[j + 1])
        {
          temp = a[j];
          a[j] = a[j + 1];
          a[j + 1] = temp;
        }
  }
}
```

Character Arrays:

Character Arrays can be constructed using the following form,

```
char[] a = new char[5];
```

Example,

```
public class Main1 {
```

```
/**
 * Converts a String to a character array
 */
public void convertStringToCharArray() {
    String str = "Abcdefg";
    char[] cArray = str.toCharArray();
    for (char c=0;c<=cArray.length-1;c++)
        System.out.println(cArray[c]);
}
public static void main(String[] args) {
    new Main1().convertStringToCharArray();
}
}
```

8.3 Two dimensional arrays

Two-dimensional arrays are defined as "an array of arrays" which contains two or more indexes. Since an array type is a first-class Java type, we can have an array of integers, an array of Strings, or an array of Objects. For example, an array of ints will have the type int[]. Similarly we can have int[][], which represents an "array of arrays of ints". Such an array is said to be a two-dimensional array.

The command

```
int[][] A = new int[3][4]
```

Declares a variable, A, of type int[][], and it initializes that variable to refer to a newly created object.for Example,

```
public class Array{
  public static void main(String[] args)   {
    int[][] a = new int[3][3];
    a[0][0] = 1;
  }
}
```

Multidimensional arrays

To store data in more dimensions a multi-dimensional array is used. The Java programming language does not really support multi-dimensional arrays. It does, however, support an array of arrays. In Java, a two-dimensional array 'x' is an array of one-dimensional array. For instance :-

```
int[][] x = new int[3][5];
```

8.4 Strings

Every string is actually an object of type string. Even string constants are actually string objects.A simple String can be created using a string literal enclosed inside double quotes as shown;

```
String str1 = “My name is babu”;
```

Since a string literal is a reference, it can be manipulated like any other String reference. The reference value of a string literal can be assigned to another String reference.

If two or more Strings have the same set of characters in the same sequence then they share the same reference in memory. Below illustrates this phenomenon.

```
String str1 = “My name is babu”;
String str2 = “My name is babu”;
String str3 = “My name ”+ “is babu”; //Compile time expression
String name = "babu";
String str4 = “My name is” + name;
String str5 = new String(“My name is babu”);
```

In the above code all the String references str1, str2 and str3 denote the same String object, initialized with the character string: “My name is babu”. But the Strings str4 and str5 denote new String objects.

8.5Vectors

Vectors differ considerably from arrays. Arrays are of only one type and the number of elements cannot be changed. Vectors instead can hold a mix of class objects (they are, of course, all subclasses of Object.) The size of a vector may increase and / or decreasedepending on the program. Vector is synchronized. Constructors for Vector class,

Vector(int, int) Constructs an empty vector with the specified storage capacity and the specified capacity Increment.
Vector(int) Constructs an empty vector with the specified storage capacity.
Vector() Constructs an empty vector.

List of few methods from Vector class,

addElement(Object) Adds the specified object as the last element of the

vector.
capacity()   Returns the current capacity of the vector.
contains(Object)   Returns true if the specified object is a value of the collection.
elementAt(int)   Returns the element at the specified index.
firstElement()   Returns the first element of the sequence.
indexOf(Object)   Searches for the specified object, starting from the first position and returns an index to it.
lastElement()   Returns the last element of the sequence.
removeAllElements() Removes all elements of the vector.
removeElementAt(int) Deletes the element at the specified index.
size()   Returns the number of elements in the vector.
toString()   Converts the vector to a string.

Example :

```
  Vector list = new Vector ();
  list.addElement (" a new string object");
  list.addElement (" another new string object");
  list.addElement (new Date ());
  list.addElement (new Date ());
  list.removeElementAt (3);
```

Example program :

```
import java.util.*;
public class VectorDemo
{
  public static void main(String[] args)
{
    Vector vector = new Vector();
    int primitiveType = 10;
    Integer wrapperType = new Integer(20);
    String str = "sunil kumar";
    vector.add(wrapperType);
    vector.add(str);
    vector.add(2, new Integer(30));
    System.out.println("the elements of vector: " + vector);
    System.out.println("The size of vector are: " + vector.size());
    System.out.println("The elements at position 2 is: " +
vector.elementAt(2));
    System.out.println("The first element of vector is: " +
vector.firstElement());
    System.out.println("The last element of vector is: " +
vector.lastElement());
    vector.removeElementAt(2);
    Enumeration e=vector.elements();
```

NOTES

```
System.out.println("The elements of vector: " + vector);
while(e.hasMoreElements()){
  System.out.println("The elements are: " + e.nextElement());
}   }}
```

Output :

```
the elements of vector: [20, sunil kumar, 30]
The size of vector are: 3
The elements at position 2 is: 30
The first element of vector is: 20
The last element of vector is: 30
The elements of vector: [20, sunil kumar]
The elements are: 20
The elements are: sunil kumar
```

8.6 Wrapper classes

A primitive wrapper class in the Java programming language is one of eight classes provided in the java.lang package to provide object methods for the eight primitive types. All of the primitive wrapper classes in Java are immutable.The primitive wrapper classes and their corresponding primitive types are:

Primitive type	Wrapper class
byte	Byte
short	Short
long	Long
float	Float
double	Double
char	Char
Boolean	Boolean

For example , Integer class has parseInt() method to parse integer from string data type.

```
Class cmdLine {
        public static void main(String args[])
        {
        int u, sum = 0;
        for(i=0;i<args.lenth;i++)
                sum += Integer.parseInt(args[i]);
        System.out.println("sum of numbers ="+sum);
}
```

The **output** is :

```
C:\java>javac cmdLine.java

C:\java>java  cmdLine 10 20 30
   Sum of numbers = 60
```

8.7 Enumerated Types

Java enums can be classes which have a fixed set of constants. These constants are final and static. They cannot be changed or modified. Enumerated data types are used to create own classes. The enum type has a values() method which returns a set of all enum constants. It can be used when we have all possible values at compile time, for the use of choices and menus etc., for example

```
enum Choice
{
   LOW,MIDDLE, HIGH
}
public lass test
{
public static void main(String[] args)
Choice c1= Choice.HIGH;
System.out.println(c1)
}
}
```

Output:

HIGH

8.8 Interfaces: Multiple Inheritances

The mechanism of inheriting the features of more than one base class into a single subclass is known as multiple inheritances.

Java does not support multiple inheritances directly but the multiple inheritances can be achieved by using the interface.

- ***In Java Multiple Inheritance can be achieved through use of Interfaces by implementing more than one interface in a class.***

8.11 Check Your Progress Questions

1. Define Array.
2. Differentiate between array and vector.

8.12 Answers to Check Your Progress Questions

1. An Array is an ordered list of elements stored in a single variable with variable indexes. The values of the array are of same type.

2. Vectors differ considerably from arrays. Arrays are of only one type and the number of elements cannot be changed. Vectors instead can hold a mix of class objects (they are, of course, all subclasses of Object.)

8.13 Summary

The java array enables the user to store the values of the same type in contiguous memory allocations. Arrays are always a fixed length abstracted data structure which cannot be altered when required. Two-dimensional arrays are defined as "an array of arrays" which contains two indexes.

Vectors differ considerably from arrays. Arrays are of only one type and the number of elements cannot be changed. Vectors instead can hold a mix of class objects (they are, of course, all subclasses of Object.)

A primitive wrapper class in the Java programming language is one of eight classes provided in the java.lang package to provide object methods for the eight primitive types.

8.14 Key Words

String is actually an object of type string. Even string constants are actually string objects.

Array is an ordered list of elements of same type.

Vectors can hold a mix of class objects (they are, of course, all subclasses of Object.)

8.15 Self-Assessment Questions and Exercises

1) Explain how to define and use arrays. Give examples
2) What are multidimensional arrays?
3) Write a java program to sort 10 given numbers.
4) Write a java program to count number of vowels using char array.

8.16 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998.

UNIT 9
PACKAGES AND INTERFACES

Structure

9.1 Introduction

Java classes and interfaces are wrapped into packages. There are some predefined packages such as java.applet, java.awt(abstract windowing tool kit) and java.io (input and output). The users can also define their own packages with set of its methods and data members. Packages provide opportunity in combing classes belong particular project or work. Packages can also further divided into sub packages.An interface is a skeleton of a class showing the methods the class with have when someone implements. Using the keyword interface you can fully abstract a class interface from its implementation.

9.2 Objectives

After going through the unit you will be able to;

- Learn about creating and importing packages
- Know about different type of access modifiers
- Understand defining, extending and implementing interfaces
- Set the class path to access packages.

9.3 Packages

Packages are a collection of classes and interfaces of similar nature. Packages are containers for classes that are used to keep the class namespace compartmentalized. For example java.io package contains classes and interfaces for various kinds of input and output.java programs automatically import all classes in the java.lang package.

9.3.1 Defining a package

To create packages simply include the package keyword as the first command in a java source file. Any classes declared within that file will belong to specified package. The package statement defines a namespace in which classes are stored.

Note: -

If you omit the package statement class names are put into default a package which has no name.

General format:-

```
package pkgname;
```

Example:-

```
package mypackage;
```

Java uses file system directories to store packages.

Example::-

The .class files for any classes you declare to be part of MyPackage must be stored in a directory called MyPackage.

Note:-
The directory name must match the package name exactly.
More than one file can include the same package statement.

You can create a hierarchy of packages. To do so simply separate each package name from the one above it by using of a period operator(.).

General format for multilevel package:-

```
package pk1[.pk2[.pk3]];
```

Example:

```
package java.awt.image;
```

The following Example explains creating and importing package called MyPack.

Create Balance.java file and save it in c:\jdk1.3\bin\MyPack directory.

```
package MyPack;

/* Now, the Balance class, its constructor, and its show() method
are public.  This means that they can be used by non-subclass code
outsidetheir package.
*/

public class Balance
{
String name;
double bal;

public Balance(String n, double b)
{
name=n;
bal=b;
}

public void show()
{
        if(bal<0)
        System.out.println("------>");
        System.out.println(name + ": $" +bal);
}
}
```

Now compile that Balance.java file.

C:\jdk1.3\bin\MyPack>javac Balance.java

When you run this file

C:\jdk1.3\bin>java MyPack.Balance

Exception in thread “main” java.lang.NoSuchMethodErro : main will occur.Because there is no main() method inside Balance class. It can be access by importing the Balance.class from another java file.

9.3.2 Importing Packages

In a java source file import statements occur immediately following the package statement and before any class definition.

General format:- import pkg1[.pkg2].(classname/*);

where, pkg → name of the top level package

pkg2→name of the subordinate package inside outer package separated by .(dot)

Note: No practical limit on depth of package hierarchy

- → to import the entire package.

For example to import the above mentioned Balance class, create TestBalance.java file and save it in c:\jdk1.3\bin> directory

```
import MyPack.*;
class TestBalance{
public static void main(String args[]){

/*Because Balance is public, you may use Balance
class and call its constructor */

Balance test = new Balance ("kumar", 99.88);

test.show(); // You may also call show()
}
}
```

Now compile this file

C:\jdk1.3\bin>javac TestBalance.java

Run this file

C:\jdk1.3\bin>java TestBalance.java

Output :

Kumar 99.88

Note that to execute TestBalance class you must one level up in the

Balance.class or you can set the classpath.

Setting Class path:

CLASSPATH variable play a significant role in locating classes. If classpath is not set, java will look for the classes in the current directory and the default directory that is generally c:\jdk\lib. If the classpath is specified then java will look only in those directories specified by the variable CLASSPATH. Remember to include current directory and the default directory also when you are setting classpath.

Access Modifiers : Access modifiers are used to specify the visibility and accessibility of a class, member variables and methods. Java provides some access modifiers like: public, private etc.. These can also be used with the member variables and methods to specify their accessibility.

- **public** keyword specifies that the public class, the public fields and the public methods can be accessed from anywhere.
- **private:**This keyword provides the accessibility only within class i.e. private fields and methods can be accessed only within the same class.
- **protected:**This modifier makes a member of the class available to all classes in the same package and all sub classes of the class.

default : When you don't write any access modifier then default is considered. It allows the class, fields and methods accessible within the package only.	**Private**	**No modifier**	**Protected**	**Public**
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non subclass	No	No	No	Yes

9.4 Interfaces

Interfaces are designed to **support dynamic method resolution at run time**. Java introduces the notion of interfaces to recover much of the

functionality that multiple inheritances give you. Java chooses this concept because:

- Multiply inheritance makes compilers either very complex or very inefficient
- With multiple inheritances there are name clashes in the base classes. That is, if tow base classes have the methods with the same name, then this results in the name clash in the class which inherits these two base classes.

Interfaces avoid this kind of problem.

Interfaces are syntactically similar to classes, but they lack instance variables and their methods are declared without any body. Once it is defined, any number of classes can implement an interface. Also one class can implement any number of interfaces.

To implement an interface, a class must create the complete set of methods defined by the interface. However, each class is free to determine the details of its own implementation.

9.4.1 Defining an Interface

The General form of an interface :

```
access interface iname{
  returntype methodname1(parameter list);
  returntype methodname2(parameter list);
  type final variablenam1 = value;
  type final variablenam2 = value;
        //..
  returntype methodnameN(parameter list);
  type final variablenamN = value;
}
```

Where, **access is either public or not used.**

Variables can be declared inside of interface declarations but they are implicitly final and static, they cannot be changed by the implementing class and must be initialized by constant value. All methods and variables are implicitly public if the interface itself is declared as public.

Example

```
        interface ifacename
  {
```

NOTES

```
void display(int param1);
}
```

9.4.2 Extending Interfaces

One interface can inherit another by used of the **keyword extends**. The general format is the same as for inheriting classes. When a class implements an interface that inherits another interface, it must provide implementations for all methods defined within the interface inheritance hierarchy.

Example:

```
interface parent{
                    void method1();
                    void method2();
}

interface child extends parent
{
      void method3();
}

// this class must implement all of  parent and child methods

class Myclass implements child
{
  public void method1(){
      System.out.println("implementation of method1");
   }
   public void method2(){
      System.out.println("implementation of method2");
   }
   public void method3(){
      System.out.println("implementation of method3");
   }
}

class Iface
{
      public static void main(String args[])
     {
       Myclass ob= new Myclass();
       ob.method1();
       ob.method2();
       ob.method3();
}
```

```
}
}
```

9.4.3 Implementing Interfaces

To implement an interface, include the *implements* keyword in a class definition and then create the methods defined by the interface. Once an interface has been defined, one or more classes can implement that interface.

The General Format of a class that implement an interface

```
access class classname[extends superclass] [implements interface [,interface…]]
{
  //  class body
}
```

Where, access is either public or not used. If a class implements more than one interface, the interfaces are separated by comma.

Example 1:

```
class myclass implements ifacename
{
     // Implement ifacename interface
      public void display (int p)
        {
        System.out.println("display is called with "+p);
        }
}
```

Example 2:

```
class myclass implements ifacename
{
     // Implement ifacename interface

      public void display (int p)
{
        System.out.println("display is called with "+p);
}

void nonIfacemeth()
```

```
{
System.out.println("Class can define its own methodstoo..")
}
```

Accessing implementations through interface references

You can declare variable as object references that use an interface rather than a class type. Any instance of any class that implements the declared interface can be stored in such a variable.

The following example calls the display() method, via an interface reference variable.

```
class ireftest
{
            public static void main(String args[])
{
                ifacename c = new myclass();
                c.display(100);
  }
  }
```

Variable in Interfaces

Variables can be declared inside interface but they are **final**. You can use interfaces to import shared constants into multiple classes.

Example ;

```
        interface constants
        {
          final int OK =1;
        final int NOTOK=2;
        final int CANCEL=3;
        }
```

9.4.4 Putting Classes Together

First, an interface can only contain abstract methods and/or static final variables (constants). Classes, on the other hand, can implement methods and contain variables that are not constants.

Second, an interface cannot implement any methods. A class that implements an interface must implement all methods defined in that interface. An interface has the ability to extend from other interfaces, and (unlike classes) can extend from multiple interfaces. Furthermore, an

interface cannot be instantiated with the new operator;

for example, Runnable a=new Runnable(); is not allowed.

9.5 Check Your Progress Questions

1. List the name of access modifiers.
2. Define Interface.
3. What are packages?

9.6 Answers to Check Your Progress Questions

1. Public, private, protected, default

2. An interface is a skeleton of a class showing the methods the class with have when someone implements.

3. Packages are a collection of classes and interfaces of similar nature.

9.7 Summary

Packages are a collection of classes and interfaces of similar nature. Packages are containers for classes that are used to keep the class namespace compartmentalized. Access modifiers are used to specify the visibility and accessibility of a class, member variables and methods. Java provides some access modifiers like: public, private etc. Interfaces are designed to support dynamic method resolution at run time.

9.8 Key Words

Access Modifiers : Access modifiers are used to specify the visibility and accessibility of a class, member variables and methods. Java provides some access modifiers like: public, private etc.

public keyword specifies that the public class, the public fields and the public methods can be accessed from anywhere.

private: This keyword provides the accessibility only within class i.e. private fields and methods can be accessed only within the same class.

protected: This modifier makes a member of the class available to all classes in the same package and all sub classes of the class.

default : When you don't write any access modifier then default is considered. It allows the class, fields and methods accessible within the package only.

Interfaces are designed to support dynamic method resolution at run time.

9.9 Self-Assessment Questions and Exercises

1) What are packages? Give predefined package names.
2) How do define a new package? Explain
3) Explain how to import newly created packages.
4) Write short note on setting classpath for accessing packages.
5) Explain the role of access modifiers in packages.
6) Define interface.
7) What are the merits of interface?
8) Describe how to define and implement interfaces.
9) How variables are defined in interface? Explain.
10) Distinguish between class and interface.

9.10 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998

BLOCK 4
MULTITHREADING, EXCEPTION AND APPLETS

UNIT 10
MULTITHREADED PROGRAMMING

Structure

10.1 Introduction

A thread is a **lightweight** process which exists within a program and executed to perform a special task. Several threads of execution may be associated with a single process. Each thread has its own local variables, program counter and lifetime.

In single threaded runtime environment, operations are executes sequentially i.e. next operation can execute only when the previous one is complete. Thus a process that has only one thread is referred to as a **single-threaded** process, while a process with multiple threads is referred to as a **multi-threaded** process.

10.2 Objectives

After going through the unit you will be able to;

- Create Threads
- Extending the thread class
- Stopping and Blocking a thread
- Understand Life cycle of a thread
- Use thread methods
- Know about Thread Exceptions
- Learn about thread Priority
- Discuss about Synchronization
- Learn how to Implement the ‘Runnable’ Interface

10.3 Creating a Thread

To create a new thread, your program will either extend **Thread** or implement the **Runnable** interface.

The Main Thread

The main() method is the first executable method runs in a one thread when java programs starts. The main thread creates some other threads called child threads. Often main() must be the last thread to finish execution because it performs various shutdown actions. If no other threads are created by the main thread, then program terminates when the main() method complete its execution. Main thread can be controlled through a **Thread** object. To do so, you must obtain a reference to it by calling the method **currentThread( )**, which is a **public static** member of **Thread**.

Its **general form** is shown here:

```
static Thread currentThread( )
```

This method returns a reference to the thread in which it is called. Once you have areference to the main thread, you can control it just like any other thread. for Example,

```
// Controlling the main Thread.
class MainThreadDemo
{
public static void main(String args[])
{
        Thread t = Thread.currentThread();
```

```
System.out.println("Current thread: " + t);
// change the name of the thread
t.setName("My Thread");
System.out.println("After name change: " + t);
try {
    for(int n = 5; n > 0; n--) {
        System.out.println(n);
        Thread.sleep(1000);}
} catch (InterruptedException e)
{
System.out.println("Main thread interrupted");
}
}
}
```

Here is the output generated by this program:

```
Current thread: Thread[main,5,main]
After name change: Thread[My Thread,5,main]
5
4
3
2
1
```

10.4 Creating a Thread by Extending Thread Class

The second way to create a thread is to create a new class that extends **Thread.**The Thread class defines Several constructors for creating new Thread instances.

```
Thread()
Thread(String)
Thread(Runnable)
Thread(Runnable,String)
Thread(ThreadGroup,String)
Thread(ThreadGroup,Runnable)
Thread(ThreadGroup,Runnable,String)
Thread(ThreadGroup, Runnable, String, long)
```

ThreadGroup– All threads belongs to an instance of the **ThreadGroup** Class. ThreadGroup is used to represent a group of threads. ThreadGroups can be shown in a hierarchical manner. There is only one root ThreadGroup that contains all other thread and groups and each subgroup can contain other groups and threads. All thread have only one thread

NOTES

group. And all thread groups (except the root thread group) belongs to exactly one parent thread group.

Steps for creating Thread by Extending Thread class

1. Extend the **java.lang.Thread** Class.
2. Override the **run( )** method in the subclass from the Thread class to define the code executed by the thread.
3. Create an **instance** of this subclass. This subclass may call a Thread class constructor by subclass constructor.
4. Invoke the **start( )** method on the instance of the class to make the thread eligible for running.

Example To create new thread by extending Thread class

```
class MyThread1 extends Thread
{
 String s=null;
 MyThread1(String s1){
  s=s1;
  start();
 }
 public void run(){
    System.out.println(s);
 }}
public class RunThread{
 public static void main(String args[]){
  MyThread1 m1=new MyThread1("Thread started....");
 }
}
```

10.5 Stopping and Blocking a thread

The Thread class has a method suspend( ) to temporarily halt the thread and resume( ) that re-starts it at the point it was halted. A program used suspend( ) and resume( ), which are methods defined by Thread, to pause and restart the execution of a thread. They have the form shown below:

```
final void suspend( )
final void resume( )
```

The Thread class also defines a method called stop( ) that stops a thread.
Its signature is shown here:

```
final void stop( )
```

Once a thread has been stopped, it cannot be restarted using resume( ).

10.6 Life cycle of a thread

The thread in java has four states in its life cycle. They are new, runnable, Non-runnable(blocked/Waiting) and terminated(dead).The figure 10.1 shows the life cycle of a thread.

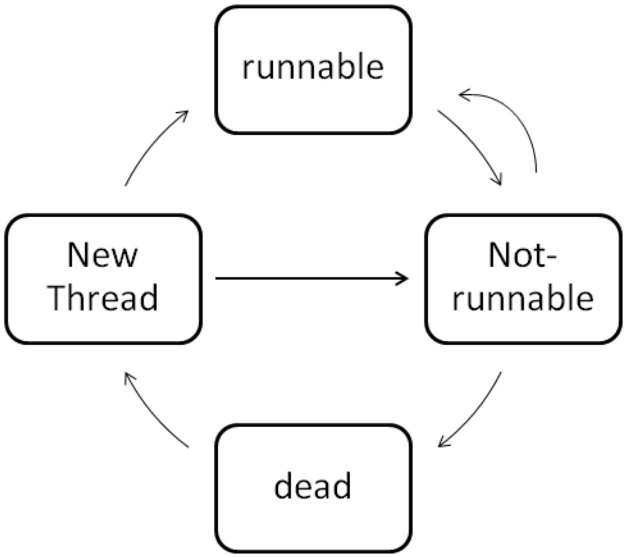


Figure 10.1 Life cycle of a thread

1. New

   When you create a new instance of a thread class and before calling the start() method, the thread state is called new.

2. Runnable

   The thread is in runnable state when the invocation of start() method.

3. Non-runnable(blocked)

   When the thread is still alive but not under running state the thread is called blocked or non-runnable state.

4. Terminated

When the run method is existed the thread goes to terminate or dead state.

10.7 Using thread methods

The **Thread** class defines several methods that help manage threads. The ones that will be used in this chapter are shown here:

Method	**Meaning**
getName()	Obtain a thread's name.
getPriority()	Obtain a thread's priority.
isAlive()	Determine if a thread is still running.
join()	Wait for a thread to terminate.
run ()	Entry point for the thread.
sleep ()	Suspend a thread for a period of time.
start ()	Start a thread by calling its run method

10.8 Thread Exception

In Java all uncaught exceptions are handled by code outside of the run() method before the thread terminates. The following example shows how to handle exception in thread.

```
class MyThreadDemo extends Thread
{
  public void run(){
    System.out.println("Throwing in " +"MyThread");
    throw new RuntimeException();
  }
}
public class MainDemo
{
  public static void main(String[] args)
{
    MyThreadDemo t = new MyThreadDemo();
    t.start();
    try {
      Thread.sleep(1000);
    } catch (Exception x) {
      System.out.println("Exception - Caught it" + x);
    }
    System.out.println("Exiting main");
```

```
  }
}
```

10.9Thread Priorities

In Java, thread scheduler can use the thread **priorities** in the form of **integer value** to each of its thread to determine the execution schedule of threads . The **thread scheduler** provides the CPU time to thread of highest priority. Priorities are integer values from 1 (lowest priority given by the constant **Thread.MIN_PRIORITY**) to 10 (highest priority given by the constant **Thread.MAX_PRIORITY**). The default priority is 5 **Thread.NORM_PRIORITY**). If two threads of the same priority are waiting for the CPU, the scheduler chooses one of them to run in a round-robin fashion.

When a Java thread is created, it inherits its priority from the thread that created it. At any given time, when multiple threads are ready to be executed, the runtime system chooses the runnable thread with the highest priority for execution.

In Java runtime system, **preemptive scheduling** algorithm is applied. If at the execution time a thread with a higher priority and all other threads are runnable then the runtime system chooses the new **higher priority** thread for execution. On the other hand, if two threads of the same priority are waiting to be executed by the CPU then the **round-robin** algorithm is applied in which the scheduler chooses one of them to run according to their round of **time-slice**.

Thread Scheduler

In the implementation of threading scheduler usually applies one of the two following strategies:

- **Preemptive scheduling** – If the new thread has a higher priority then current running thread leaves the runnable state and higher priority thread enter to the runnable state.

- **Time-Sliced (Round-Robin) Scheduling** – A running thread is allowed to be execute for the fixed time, after completion the time, current thread indicates to the another thread to enter it in the runnable state.

To set a thread's priority, use the **setPriority( )** method, which is a member of **Thread**.

General format is ,

final void setPriority(int *level*)

Here, *level* specifies the new priority setting for the calling thread. The value of *level* must be within the range **MIN_PRIORITY** and **MAX_PRIORITY**. These priorities are defined as **final** variables within **Thread**.

You can obtain the current priority setting by calling the **getPriority( )** method of **Thread.**

General Format is,

final int getPriority( )

The following example demonstrates two threads at different priorities, which do not run on a preemptive platform in the same way as they run on a nonpreemptive platform. One thread is set two levels above the normal priority, as defined by **Thread.NORM_PRIORITY**, and the other is set to two levels below it. The threads are started and allowed to run for ten seconds. Each thread executes a loop, counting the number of iterations. After ten seconds, the main thread stops both threads. The number of times that each thread made it through the loop is then displayed.for Example,

```
// Demonstrate thread priorities.
class clicker implements Runnable {
  double click = 0;
  Thread t;
  private volatile boolean running = true;
  public clicker(int p)
  {
          t = new Thread(this);
           t.setPriority(p);
  }
public void run()
{
  while (running)
  { click++; }
}
public void stop()
{ running = false; }
public void start() {
  t.start(); }
}
class HiLoPri {
public static void main(String args[])
{
  Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
```

```
   clicker hi = new clicker(Thread.MAX_PRIORITY);
  clicker lo = new clicker(Thread.MIN_PRIORITY);
lo.start();
hi.start();
try {
   Thread.sleep(10000);
} catch (InterruptedException e) {
System.out.println("Main thread interrupted.");
}
lo.stop();
hi.stop();
// Wait for child threads to terminate.
try {
hi.t.join();
lo.t.join();
} catch (InterruptedException e)
{ System.out.println("InterruptedException caught"); }
System.out.println("Low-priority thread: " + lo.click);
System.out.println("High-priority thread: " + hi.click);
}
}
```

The output of this program

```
Low-priority thread: 1.1183645553E9
High-priority thread: 1.155930947E9
```

Of course, the exact output produced by this program depends on the speed of your CPU and the number of other tasks running in the system.

10.10 Synchronization

The threads are executed independently to each other. These types of threads are called as **asynchronous threads**. But there are two problems may be occurring with asynchronousthreads.

- Two or more threads share the same resource (variable or method) while only one of them can access the resource at one time.
- If the producer and the consumer are sharing the same kind of data in a program then either producer may produce the data faster or consumer may retrieve an order of data and process it without its existing.

Suppose, you have created two methods as **increment( )** and **decrement()**. which increases or decreases value of the variable **"count"** by 1 respectively shown as:

NOTES

```
        public void increment( ) {
            count++;
        }
        public void decrement( ) {
            count--;
        }
       public int value() {
            return count;
        }
```

When the two threads are executed to access these methods (one for **increment( )**,another for **decrement( )**) then both will share the variable **"count"**. in that case, you can't be sure that what value will be returned of variable "count".

To avoid this problem, Java uses **monitor** also known as **“semaphore”** to prevent data from being corrupted by multiple threads by a keyword **synchronized** to synchronize them and intercommunicate to each other. It is basically a mechanism which allows two or more threads to share all the available resources in a sequential manner. Java's synchronized is used to ensure that only one thread is in a **critical region**. critical region is a lock area where only one thread is run (or lock) at a time. Once the thread is in its critical section, no other thread can enter to that critical region. In that case, another thread will has to wait until the current thread leaves its critical section.

When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called ***synchronization***.

You can synchronize your code in either of two ways. Both involve the use of the**synchronized** keyword, and both are examined here.

1. Synchronized Methods

2. Synchronized Blocks (Statements)

Using Synchronized Methods

Any method is specified with the keyword **synchronized** is only executed by one thread at a time. If any thread wants to execute the synchronized method, firstly it has to an object’s monitor, just call a method that has been modified with the **synchronized** keyword. While a thread is inside a synchronized method, all other threads that try to call it on the same instance have to wait. To exit the monitor and relinquish control of the object to the next waiting thread.

//This is **not Synchronized**

```
class DisplayMessage {
void display(String msg) {
        System.out.print("[" + msg);
        try {
                Thread.sleep(1000);
        } catch(InterruptedException e)
         {
        System.out.println("Interrupted");  }
        System.out.print("]"); }
        }
class Caller implements Runnable {
        String msg;
        DisplayMessage target;
        Thread t;
        public Caller(DisplayMessage targ, String s) {
                target = targ;   msg = s;
                t = new Thread(this);  t.start();
        }
public void run() {
        target.display(msg); }
}
class Sync {
public static void main(String args[]) {
        DisplayMessage target = new DisplayMessage();
        Caller ob1 = new Caller(target, "This");
        Caller ob2 = new Caller(target, "is not");
        Caller ob3 = new Caller(target, "Synchronized");
        try {
                ob1.t.join();
                ob2.t.join();
                ob3.t.join();
        } catch(InterruptedException e) {
        System.out.println("Interrupted");
  } } }
```

Output

```
[This[is not[Synchronized]]]
```

To fix the preceding program, you must *serialize* access to display**( )**. That is, you must restrict its access to only one thread at a time. To do this, you simply need to precede display**( )**'s definition with the keyword **synchronized**, as shown here:

```
class Display Message{
        synchronized void display(String msg) {
...
```

NOTES

This prevents other threads from entering display**( )** while another thread is using it.

```
/* This is Synchronized
class DisplayMessage
{
synchronized void display(String msg)
{
    System.out.print("[" + msg);
    try {
            Thread.sleep(1000);
    } catch(InterruptedException e) {
System.out.println("Interrupted"); }
    System.out.print("]"); }
}
class Caller implements Runnable
{
    String msg;
    DisplayMessage target;
    Thread t;
    public Caller(DisplayMessage targ, String s) {
            target = targ;
            msg = s;
            t = new Thread(this);
            t.start();
    }
        public void run() {
    target.display(msg);  }
}
class Sync {
public static void main(String args[])
{
    DisplayMessage target = new DisplayMessage();
    Caller ob1 = new Caller(target, "This");
    Caller ob2 = new Caller(target, "is");
    Caller ob3 = new Caller(target, "Synchronized");
    try {
            ob1.t.join();
            ob2.t.join();
            ob3.t.join();
    } catch(InterruptedException e)
{ System.out.println("Interrupted");
} } }
```

After **synchronized** has been added to display(), the output of the program is as follows:

```
[This][is][Synchronized]
```

The synchronized Statement

Imagine that you want to synchronize access to objects of a class that was not designed for multithreaded access. That is, the class does not use **synchronized** methods. How can access to an object of this class be synchronized? You simply put calls to the methods defined by this class inside a **synchronized** block.

This is the general form of the **synchronized** statement:

```
synchronized (object)
{
        // statements to be synchronized
}
```

Here, *object* is a reference to the object being synchronized. A synchronized block ensures that a call to a method that is a member of *object* occurs only after the current thread has successfully entered *object*'s monitor.

Here is an alternative version of the preceding example, using a synchronized block within the **run( )** method:

```
// This program uses a synchronized block.

class Displaymessage
{
    void display(String msg)
    {
    System.out.print("[" + msg);
    try
    {
        Thread.sleep(1000);
     } catch (InterruptedException e)
  {
    System.out.println("Interrupted");
  }
System.out.println("]");
}
}
class Caller implements Runnable
{
String msg;
Displaymessage target;
Thread t;
public Caller(Displaymessage targ, String s)
```

NOTES

```
{
        target = targ;
        msg = s;
        t = new Thread(this);
        t.start();
}
// synchronize calls to call()
public void run()
{
        synchronized(target) { // synchronized block
        target.display(msg);
}
}
}
class SynchBlockDemo
{
public static void main(String args[])
{
        Displaymessage target = new Displaymessage();
        Caller ob1 = new Caller(target, "This");
        Caller ob2 = new Caller(target, "is");
        Caller ob3 = new Caller(target, "Synchronized");
        Caller ob4 = new Caller(target, ”Example”);
// wait for threads to end
try {
        ob1.t.join();
        ob2.t.join();
        ob3.t.join();
        ob4.t.join();

} catch(InterruptedException e)
{
  System.out.println("Interrupted");
}
}
}
```

The **synchronized** statement is used inside **Caller**'s **run( )** method.

The output of the program is as follows:

[This][is][Synchronized][Example]

Messaging

After you divide your program into separate threads, you need to define how they will communicate with each other. When programming with most other languages, you must depend on the operating system to establish communication between threads. This, of course, adds overhead.

By contrast, Java provides a clean, low-cost way for two or more threads to talk to each other, via calls to predefined methods that all objects have. Java's messaging system allows a thread to enter a synchronized method on an object, and then wait there until some other thread explicitly notifies it to come out.

Inter-Thread Communication

Java provides a very efficient way through which multiple-threads can communicate with each-other. This way reduces the CPU's idle time i.e. A process where, a thread is paused running in its critical region and another thread is allowed to enter (or lock) in the same critical section to be executed. This technique is known as **Inter-thread communication** which is implemented by some methods.

These methods are defined in "**java.lang**" package and can only be called within synchronized code shown as:

```
class Shared
{
        int num=0;
        boolean value = false;
        synchronized int get() {
          if (value==true)
          try {
            wait();
            }
  catch (InterruptedException e)
  {
  System.out.println("InterruptedException caught");
  }
System.out.println("consume: " + num);
value=false;
notify();
return num;
}

synchronized void put(int num)
{
  if (value==true)
  try {
    wait();
    }
  catch (InterruptedException e)
  {
     System.out.println("InterruptedException caught");
    }
    this.num=num;
    System.out.println("Produce: " + num);
    value=false;
```

```
    notify();
    }
    }



class Producer extends Thread {
  Shared s;
  Producer(Shared s) {
    this.s=s;
    this.start();
  }
  public void run() {
    int i=0;
   s.put(++i);
    }
}

class Consumer extends Thread
{
  Shared s;
  Consumer(Shared s)
{
    this.s=s;
    this.start();
  }
  public void run() {
    s.get();
    }
}

public class InterThreadComm
{
  public static void main(String[] args)
  {
    Shared s=new Shared();
    new Producer(s);
    new Consumer(s);
  }}
```

Output:

Produce : 1
Consumes :1

In this program, two threads "**Producer**" and "**Consumer**" share the **synchronized** methods of the class **"Shared"**. At time of program execution, the **"put( )"** method is invoked through the **"Producer"** class which increments the variable **"num"** by **1**. After producing 1 by the producer, the method **"get( )"** is invoked by through the **"Consumer"** class which retrieves the produced number and returns it to the output. Thus the Consumer can't retrieve the number without producing of it.

wait() method

- the **wait()** method causes a thread to release the lock it is holding on an object; allowing another thread to run
- the **wait()** method is defined in the **Object** class
- **wait()** can only be invoked from within **synchronized** code
- it should **always** be wrapped in a **try** block as it throws IOExceptions
- there are actually three **wait()** methods
    1. wait()
    2. wait(long timeout)
    3. wait(long timeout, int nanos)
- the **timeout** is measured in **milliseconds**
- **nanos** is measured in **nanoseconds**
- wait() can only invoked by the thread that own's the lock on the object
- when wait() is called, the thread becomes disabled for scheduling and lies dormant until one of four things occur:

    1. another thread invokes the **notify()** method for this object and the scheduler arbitrarily chooses to run the thread
    2. another thread invokes the **notifyAll()** method for this object
    3. another thread **interrupts** this thread
    4. the specified wait() time elapses
- when one of the above occurs, the thread becomes re-available to the Thread scheduler and competes for a lock on the object
- once it regains the lock on the object, everything resumes **as if** no suspension had occurred
- if the thread was **interrupted** by another thread, an InterruptedException is thrown BUT not until after the thread regains it's lock on the object

notify() and notifyAll() Methods

- the **notify()** and **notifyAll()** methods are defined in the Object class
- they can only be used within **synchronized** code
- notify() wakes up a single thread which is waiting on the object's lock
- if there is more than one thread waiting, the choice is arbitrary i.e. there is no way to specify which waiting thread should be re-awakened
- **notifyAll()** wakes up ALL waiting threads; the scheduler decides which one will run
- if there are no waiting threads, the notify’s are forgotten
- only notifications that occur **after** a thread has moved to wait state will effect it; earlier notifies are irrelevant

Deadlock

A situation where a thread is waiting for an object lock that holds by second thread, and this second thread is waiting for an object lock that holds by first thread, this situation is known as **Deadlock**.

NOTES

Suspending, Resuming, and Stopping Threads

The **Thread** class has a method **suspend( )** to temporarily halt the thread and **resume( )** that re-starts it at the point it was halted. A program used **suspend( )** and **resume( )**, which are methods defined by **Thread**, to pause and restart the execution of a thread. They have the form shown below:

```
final void suspend( )
final void resume( )
```

The **Thread** class also defines a method called **stop( )** that stops a thread. Its signature is shown here:

```
final void stop( )
```

Once a thread has been stopped, it cannot be restarted using **resume( )**.

10.11 Creating a Thread by Implementing Runnable Interface

Steps for creating Thread by implementing runnable interface

1. A Class implements the **Runnable** Interface, override the run() method to define the code executed by thread. An object of this class is Runnable Object.
2. Create an object of **Thread** Class by passing a Runnable object as argument.
3. Invoke the **start( )** method on the instance of the Thread class.

To implement **Runnable**, a class need only implement a single method called **run( )**, which is declared like this:

```
public void run( )
```

Inside **run( )**, you will define the code that constitutes the new thread. It is important to understand that **run( )** can call other methods, use other classes, and declare variables, just like the main thread can. The only difference is that **run( )** establishes the entry point for another, concurrent thread of execution within your program. This thread will end when **run( )** returns.

After you create a class that implements **Runnable**, you will instantiate an object oftype **Thread** from within that class. **Thread** defines several constructors. The one that you will use is shown here:

Thread(Runnable *threadOb*, String *threadName*)

In this constructor, *threadOb* is an instance of a class that implements the **Runnable** interface. This defines where execution of the thread will begin. The name of the new thread is specified by *threadName*.

After the new thread is created, it will not start running until you call its **start( )** method, which is declared within **Thread**. In essence, **start( )** executes a call to **run( )**.

The **start( )** method is shown here:

```
void start( )
```

Example : To create new thread by implementing Runnable Interface

```
class MyThread1 implements Runnable
{
  Thread t;
  String s=null;
  MyThread1(String s1)
{
    s=s1;
   t=new Thread(this);
   t.start();
  }
  public void run()
{
     System.out.println(s);
    }
}
public class RunableThread
{
  public static void main(String args[])
{
   MyThread1 m1=new MyThread1("Thread started....");
    }
}
```

10.12 Check Your Progress Questions

1) What are various states of a Thread? Explain.
2) What is synchronization?

10.13 Answers to Check Your Progress Questions

1. runnable, new, not-runnable, terminated
2. When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time. The process by which this is achieved is called **synchronization**.

10.14 Summary

A thread is a lightweight process which exists within a program and executed to perform a special task. It has four states namely new, runnable, not-runnable and terminated. Threads can be implemented by extending thread class and implementing runnable interface. Thread has various priorities such as normal, maximum and minimum.

10.15 Key Words

- **A thread** is a lightweight process which exists within a program and executed to perform a special task
- **Deadlock** A situation where a thread is waiting for an object lock that holds by second thread, and this second thread is waiting for an object lock that holds by first thread, this situation is known as Deadlock.

10.16 Self-Assessment Questions and Exercises

1) What is multitasking?
2) Define thread.
3) Write short note on main() thread.
4) Explain the ways to create a new thread. Explain.
5) What is multithreading?
6) What are the merits of multithreading?
7) Write a java program to illustrate multithreading.
8) Write short note on isAlive() and join() methods.
9) What are Thread Priorities? Explain how to assign priorities to threads.
10) Write shot note on thread scheduler.
11) Briefly explain how synchronization is achieved in java.
12) Write short note on Messaging.

13) Briefly explain the concept of inter-thread communication
14) Write short note on wait(),notify() and notifyAll() methods.
15) Define deadlock
16) How to suspend résumé and stop threads? Explain.

10.17 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998

UNIT 11
MANAGING ERROR, EXCEPTIONS AND GRAPHICS PROGRAMMING

Structure

11.1 Introduction

Writing an error-free code is a concern of programmers. The presence of errors leads to undesirable results. To avoid unwanted termination of program execution, exception handling is required to identify and handle errors. This unit will discuss about various types of errors and the ways to handle exceptions.

11.2 Objectives

After going through the unit you will be able to;

- Learn various types of errors
- Understand exception handling mechanism
- Name some common names of exceptions
- Know about graphics programming

11.3 Types of Errors

Though it is the dream of every programmer to write error –free programs, it is not so normally. This is because the programmer has not anticipated all possible situations that might occur while running the program. The errors might be due to a programming mistake, bad input data, corrupted files etc. It is necessary to take care about these situations. Hence java introduces Exception Handling. Errors are generally classified into compile time error and run-time errors.

11.4 Exceptions

An ***exception*** is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred. Creating an exception object and handing it to the runtime system is called *throwing an exception*. After a method throws an exception, the runtime system attempts to find something to handle it. Java programs have the following advantages.

1. It helps to separate the error handling code from the regular source code.
2. It prevents the program from automatically terminating.
3. The run time system searches backwards through the call stack, beginning with the method in which the error occurred, until it finds a method that contains an appropriate exception handler.

A Java exception is an object. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. They are instances of classes that inherit from the class called **Throwable.**

Exception Types

All exception types are subclasses of the built-in class **Throwable**. Thus, **Throw able**is at the top of the exception class hierarchy. **Throwable** are two subclasses.

```
                   Object
                     |
                   Throwable
                     |
           ---------------------------
           |                         |
         Error                   Exception
                                     |
                              RuntimeException
                                     |
                              -----------------------
                              |   |   |   |   |  |
```

One is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. This is also the class that you will subclass to create your own custom exception types. There is an important subclass of **Exception**, called **RuntimeException**. Exceptions of this type are automatically defined for the programs that you write and include things such as division by zero and invalid array indexing.

The other is **Error**, which defines exceptions that are not expected to be caught under normal circumstances by your program. Exceptions of type **Error** are used by the Java run-time system to indicate errors having to do with the run-time environment, itself. This unit will dealing with exceptions of type **Exception.**

RuntimeExceptions are those exceptions that occur with in the java runtime system. This includes arithmetic exceptions(such as when dividing by zero).Runtime exceptions can occur anywhere in a program and in a typical program can be very numerous.

Java exception handling is managed via five keywords: They are **try**, **catch**, **throw**, **throws**, and **finally**.

- Program statements that you want to monitor for exceptions are contained within a **try** block. If an exception occurs within the **try** block, it is thrown. Your code can catch this exception (using **catch**) and handle it in some rational manner.

- To manually throw an exception, use the keyword **throw**.

- Any exception that is thrown out of a method must be specified as such by a **throws** clause.

- Any code that absolutely must be executed before a method returns is put in a **finally** block.

11.4.1 Syntax of Exception Handling code

This is the general form (syntax) of an exception-handling block:

```
try
{
        // block of code to monitor for errors
}
catch (ExceptionType1 exOb) {
        // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
        // exception handler for ExceptionType2
}
        // ...
finally
{
        // block of code to be executed before try block ends
}
```

Here, *ExceptionType* is the type of exception that has occurred.

Using try and catch

To guard against and handle a run-time error, simply enclose the code that youwant to monitor inside a **try** block. Immediately following the **try** block, includes a **catch** clause that specifies the exception type that you wish to catch.

Syntax :

```
try{
    ………
    ………
}
catch(<exceptionclass1> <obj1>){
    ………
    ………
}
```

For Example,

```
class Excep {
public static void main(String args[]) {
int d, a;
try { // monitor a block of code.
        d = 0;
        a = 42 / d;
        System.out.println("This will not be printed.");
      }catch (ArithmeticException e) { // catch divide-by-zero
error
        System.out.println("Division by zero.");
}
System.out.println("After catch statement.");
}
}
```

Output :

```
Division by zero.
After catch statement.
```

Notice that the call to **println( )** inside the **try** block is never executed. Once an exception is thrown, program control transfers out of the **try** block into the **catch** block. Once the **catch** statement has executed, program control continues with the next line in the program following the entire **try**/**catch** mechanism.

11.4.2 Multiple Catch statements

There may be situations where more than one exception could be raised by a single piece of code. To handle this type of situation, you can specify two or more **catch** clauses, each catching a different type of exception.

Syntax

```
try{
………
………
}
catch(<exceptionclass_1> <obj1>)
{
//statements to handle the exception
}
catch(<exceptionclass_2> <obj2>)
```

```
{
//statements to handle the exception
}
catch(<exceptionclass_N> <objN>)
{
//statements to handle the exception
}
```

For Example,

```
public class Multi_Catch
{
    public static void main (String args[])
      {
       int array[]={20,10,30};
       int num1=15,num2=0;
       int res=0;
     try
     {
        res = num1/num2;
       System.out.println("The result is" +res);
       for(int ct =2;ct >=0; ct--)
          System.out.println("The value of array are" +array[ct]);
       }catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error…. Array is out of Bounds");
       }catch (ArithmeticException e) {
                   System.out.println ("Can't be divided by Zero");
        }
     }
   }
```

Output

```
Can't be divided by Zero
```

11.4.3 Using finally statement

Other than exception handling the finally clause helps you in avoiding any cleanup code accidentally bypassed by a return etc. For example, if a method opens a file upon entry and closes it upon exit, then you will not want the code that closes the file to be bypassed by the exception-handling mechanism. It is always a good practice to use finally clause after the try and catch block because the finally block always executes even if an unexpected exception occurs i.e. whether or not an exception thrown. The

finally block executes if and only if the try block exits. The statements within the finally block gets executed by the runtime system without taking care of what happens within the try block.

> **Note:** The **finally** block will execute whether or not an exception is thrown.
> The **finally** clause is optional.
> Each **try** statement requires at least one **catch** or a **finally** clause.

Finally block can be useful for closing file handles and freeing up any other resources that might have been allocated at the beginning of a method with the intent of disposing of them before returning.

Syntax:

```
try{
      ………
      ………
}
catch(<exceptionclass1> <obj1>){
      ………
      ………
}
finally{
      ………
      ………
}
```

The following example, whether the exception is occurred or not always the finally statement is executed and its closes the file.

```
import java.io.*;
class Test{
public static void main(String args[])throws IOException {
FileInputStream fis=null;
try{
    fis = new FileInputStream (new File (args[0]));
}
catch (FileNotFoundException e){
System.out.println("File not found!");
}
finally{
    fis.close();}
}
}
```

11.4.4 Throwing our own Exceptions

It is possible for your program to throw an exception explicitly, using the **throw** statement.

The general form of **throw** is shown here:

throw *ThrowableInstance*;

Here, *ThrowableInstance* must be an object of type **Throwable** or a subclass of**Throwable**. Simple types, such as **int** or **char**, as well as non-**Throwable** classes, such as **String** and **Object**, cannot be used as exceptions.

Here is a sample program that creates and throws an exception. The handler thatcatches the exception rethrows it to the outer handler.

```
// Demonstrate throw.
class ThrowDemoprogram
{
static void demoproc() {
try {
     throw new NullPointerException("demo");
} catch(NullPointerException e)
 {
 System.out.println("Caught inside demoproc.");
 throw e; // rethrow the exception
 }
}
public static void main(String args[])
{
try {
     demoproc();
} catch(NullPointerException e)
{
 System.out.println("Recaught: " + e);
}
}
}
```

Output:

```
Caught inside demoproc.
Recaught: java.lang.NullPointerException: demo
```

Using 'throws'

'throws' clause is used when a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. This is the general form of a method declaration that includes a **throws** clause:

```
type method-name(parameter-list) throws exception-list{
        // body of method
}
```

Here, *exception-list* is a comma-separated list of the exceptions that a method can throw.

Example :

```
class ThrowsDemo {
static void throwOne() throws IllegalAccessException {
        System.out.println("Inside throwOne.");
        throw new IllegalAccessException("demo");
}
public static void main(String args[]) {
try {
        throwOne();
} catch (IllegalAccessException e) {
System.out.println("Caught " + e); }
}
}
```

output generated by running this example program:

inside throwOne
caught java.lang.IllegalAccessException: demo

Difference between throw and throws keywords

Whenever you want to force an exception then you use **throw**keyword. The **throw** keyword (note the singular form) is used to force an exception. It can also pass a custom message to your exception handling module. Moreover **throw** keyword can also be used to pass a custom message to the exception handling module i.e. the message which you want to be printed. For instance in the above example you have used -

throw new MyException ("can't be divided by zero");

Whereas when you know that a particular exception may be thrown or to pass a possible exception then you use **throws** keyword. Point to note here is that the Java compiler very well knows about the exceptions thrown by

some methods so it insists us to handle them. You can also use **throwsclause** on the surrounding method instead of **try and catch exception handler.**

Creating custom defined (Your Own) Exception Subclasses

You can code a class that defines an exception that is more appropriate and that mechanism of handling exception is called **Custom** or **User Defined Exception**. Just define a subclass of **Exception** (which is, of course, a subclass of **Throwable**). There are two primary use cases for a custom exception.

- Your code can simply throw the custom exception when something goes wrong.
- You can wrap an exception that provides extra information by adding your own message.

Example

```
import java.io.*;
import java.util.*;
class MyException extends Exception{
    private String nm="";
        public String getMessage(String s){
          nm=s;
         return ("you are not permitted to enter inside "+nm);
         }
}
public class ExcepDemo        {
public static void main(String args[])throws MyException,IOException
{
        String temp="";
     try
      {
         String str="kumar";
         System.out.println("Enter your name");
         BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
          temp=br.readLine();
          if(!temp.equals(str))
            throw new MyException();
         else
         System.out.println("Welcome to Java world kumar");
      }
       catch(MyException e)
{
        System.err.println(e.getMessage(temp));
```

```
            }
        catch(Exception e)
{
          System.err.println(e);       }
    }
   }
```

Output :

```
Enter your name
senthil
you are not permitted to enter inside senthil
```

In this example you have created own exception class as **MyException** that throws an exception and a function with argument as **getMessage** that shows an exception message, if the user tries to enter the another name which doesn't match with a particular predefined name. After throwing exception the control will be transferred in the catch block to handle the exception, where the function is invoked to display the message included in that function.

11.4.5 Using exceptions for Debugging

We can use exception for debugging and identifying bugs.Onecanthrow Exception from code catch it and print stack trace this way. We can ensure that program control is reaching till which point in my code and where is actual problem.

11.5 Graphics Programming

Java programming provides various functionalities to perform graphics applications. There are so many ways to create graphics. The simple way is to use java.awt.Graphics and java.awt.Canvas.Canvas is an rectangular area where the pictures are drawn.

11.5.1 The Graphics Class

The java.awt.Graphics class provides methods to draw various graphics and manage fonts and colors. It is the abstract super class for all graphics. The following are example few methods of graphics class.

drawLine(x1,y1,x2,y2) method is used to draw a line from (x1,y1) to (x2,y2).

drawRect(x1,y1,width,height) draws a rectangle.

fillRect(x1,y1,width,height) draws a filled rectangle.

drawOval(x1,y1,width,height) draws an oval shape.

fillOval(x1,y1,width,height) draws a filled Oval.

setBackground(c) sets the background color,

setColor(c) sets the text color

11.5.2 Drawing Lines, Rectangle, Circles, Ellipses, Arcs and Polygons

The java.awt.Graphics class provides various methods for displaying output primitives such as lines, rectangle, circle, ellipse, arcs and polygons.

For example;

```
import java.awt.*;
import java.applet.*;
public class drawShapes extends Applet
{

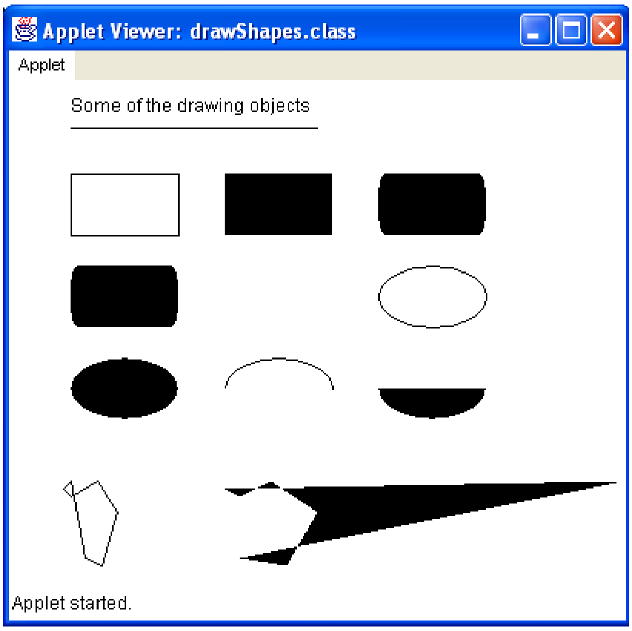
// this is for drawPolygon
int xs[] = {40,49,60,70,57,40,35};
int ys[] = {260,310,315,280,260,270,265};

// this is for fillPolygon
int xss[] = {400,150,180,200,170,150,140};
int yss[] = {260,310,315,280,260,270,265};

public void paint(Graphics g)
{
   g.drawString("Some of the drawing objects", 40,20);
   g.drawLine(40,30,200,30);
   g.drawRect(40,60,70,40);
   g.fillRect(140,60,70,40);
   g.fillRoundRect(240,60,70,40,10,20);
   g.fillRoundRect(40,120,70,40,10,20);
   g.drawOval(240,120,70,40);
   g.fillOval(40,180,70,40);
   g.drawArc(140,180,70,40,0,180);
```

```
        g.fillArc(240,180,70,40,0,-180);
        g.drawPolygon(xs,ys,7);
        g.fillPolygon(xss,yss,7);
    }
    }
```

Output



11.5.3 Line Graphs

Line graph is a graphic which has a series of points connected based on origin and (x,y) coordinates.One axis of the chart plots categories and the other axis represents the value scale. It is used to visualize the value with time or other variables. Java.awt. packages provides methods to support drawing line graphs.

11.5.4 Using Control Loops in Applets

Control loops such as for,while and do-while are statements which are used to control the statements in applets. For example the following program illustrate the use of for loop to display welcome applet four times.

```
import java.awt.*;
import java.applet.*;
/* <applet code="TestApplet" width=200 height=60>
</applet> */
```

```
public class TestApplet extends Applet {
public void paint(Graphics g) {
        for (int i=1;i<=4;i++){
          g.drawString("Welcome to Applet", 20, 20);

}}
}
```

11.5.5 Drawing Bar Charts.

A bar chart represents quantitative information. The chart consists of horizontal bars of equal width with lengths proportional to the values they represent, something that aids in instant comparison of data. One axis of the chart plots categories and the other axis represents the value scale. For example.

```
import java.applet.*;
import java.awt.*;

/*
<applet code="Bargraph.class" width =500 height =400>
<param name="time1" value ="2">
<param name="time2" value ="3">
<param name="time3" value ="4">
<param name="time4" value ="5">
<param name="time5" value ="6">
<param name="temperature1" value ="35">
<param name="temperature2" value ="36">
<param name="temperature3" value ="40">
<param name="temperature4" value ="39">
<param name="temperature5" value ="38">
</applet>
/*

public class Bargraph extends Applet
{
        int n;
        String ttime[];
        int value[];

        ttime[0] = getParameter("time1");
        ttime[1] = getParameter("time2");
        ttime[2] = getParameter("time3");
        ttime[3] = getParameter("time4");
        ttime[4] = getParameter("time5");
```

```
        value[0]= Integer.parseInt(getParameer("temperature1"));
        value[1]= Integer.parseInt(getParameer("temperature2"));
        value[2]= Integer.parseInt(getParameer("temperature3"));
        value[3]= Integer.parseInt(getParameer("temperature4"));
        value[4]= Integer.parseInt(getParameer("temperature5"));
}

public void paint(Graphics g)
{
        Font font = new Font("Arial",Font.BOLD,15);
        g.setFont(font);

for(int i=0;i<n;i++)
{
        g.setColor(Color.BLUE);
        g.drawString(year[i],20,i*50+30);
        g.setColor(Color.RED);
        g.fillRect(70,i*50+10,value[i],40);
        g.drawString(String.valueOf(value[i]+"%",180,i*50+35);
}

        String msg="Bar Chart";
        g.setColor(Color.darkGray);
        font=new Font("Arial",Font.BOLD,20);
        g.setFont(font);
        g.drawString(msg,50,300);
}
}
```

11.6 Check Your Progress Questions

1) What is an Exception?

2) List predefined exceptions.

11.7 Answers to Check Your Progress Questions

1. An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

2. Arithmetic Exception, IndexOutofBoundsException and IllegalAccessException

11.8 Summary

Errors are generally classified into compile time error and run-time errors. An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.

When an error occurs within a method, the method creates an object and hands it off to the runtime system.

11.9 Key Words

- **Throws –** This is used when a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception.

- **finally -** This block can be useful for closing file handles and freeing up any other resources that might have been allocated at the beginning of a method with the intent of disposing of them before returning.

11.10 Self-Assessment Questions and Exercises

1) What are the advantages of handling exception?
2) What are the types of exceptions?
3) Write a simple program to illustrate exception handling in java using try and catch.
4) What is the use of finally clause? Give examples
5) Write short note on uncaught exceptions
6) Briefly explain about nested try statements.
7) What is the use of throw keyword? Explain with suitable example.
8) What is the use of throws keyword? Explain with suitable example.
9) Distinguish between throw and throws keyword.

11.11 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.

NOTES

3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998

UNIT 12
APPLET PROGRAMMING

Structure

12.1 Introduction

Java implements two types of programs namely console operated program(applications) which discussed so far and applet program. Applet is a small program typically embedded with in a web page to create dynamic and interactive web applications and runs under any java enabled browser. Java applets have various states such as init, start, stop, paint and destroy. This chapter you will be able to learn about applet fundamentals.

12.2 Objectives

After going through the unit you will be able to;

- Learn how to write applet programs
- Understand the difference between applets and other applications

NOTES

- Know the life cycle of an applet
- Design a web page to invoke applets
- Learn the ways of executing applets

12.3 How applets differ from Applications

Two types of Programs you can do with java. They are application programs and Applet programs. All of the preceding examples in this book have been Java applications... Another type of program is the applet. Applets are applications that are accessed on an Internet server. An applet is a java program that can be embedded in a web page. Applets are run on any browser that supports java.

Java Applets has certain limitations,

- They cannot load or run any program stores on the user's system.
- They cannot read or write files on the user's system

12.4 Preparing to write applets

An applet displays information on the screen by using paint () method. This method is from java.awt.Component class. This method takes an instance of the Graphics class. Graphics class provides various methods to display information.

NOTE

1. All applets are subclasses of the Applet class in the java.applet.package.
2. All applets must be declared public.
3. Applets do not begin execution at **main ( )**.
4. Applets do not have main() method
5. User I/O is not accomplished with Java's stream I/O classes. Instead, Applets use the interface provided by the AWT.

12.5 Building an Applet code

Let's begin with the simple applet shown here:

```
import java.awt.*;
import java.applet.*;
public class TestApplet extends Applet {
```

```
public void paint(Graphics g) {
        g.drawString("Welcome to Applet", 20, 20);
}
}
```

This applet begins with two **import** statements. The first imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user through the AWT, not through the console-based I/O classes. The second **import** statement imports the **applet** package, which contains the class **Applet**.

The next line in the program declares the class **TestApplet**. This class must bedeclared as **public**, because it will be accessed by code that is outside the program.Inside **TestApplet**, **paint( )** is declared. This method is defined by the AWTand must be overridden by the applet. **paint( )** is called each time that the appletmust redisplay its output. Whatever the cause, whenever the applet must redraw its output, **paint( )** is called. The **paint( )** method has one parameter of type **Graphics**.

Inside **paint( )** is a call to **drawString( )**, which is a member of the **Graphics** class.This method outputs a string beginning at the specified X,Y location. It has thefollowing general form:

void drawString(String *message*, int *x*, int *y*)

Here, *message* is the string to be output beginning at *x*,*y*. In a Java window, the upper-leftcorner is location 0,0. The call to **drawString( )** in the applet causes the message "A Simple Applet" to be displayed beginning at location 20,20.Instead, an applet begins execution when the name of its class is passed to an applet viewer or to a network browser.

After you enter the source code for **TestApplet**, compile in the same way thatyou have been compiling programs. However, running **TestApplet** involves adifferent process.

12.6 Applet Life Cycle

In java, java.applet.Applet class has four methods defines the lifecycle of the servlets, They are,

```
public void init();
public void start();
public void stop();
public void destroy();
```

NOTES

The **init()** method is called exactly once in an applet's life, when the applet is first loaded. It's normally used to read PARAM tags, start downloading any other images or media files you need, and set up the user interface.

The **start()** method is called at least once in an applet's life, when the applet is started or restarted. A start() method is often used to start any threads the applet will need while it runs.

The **stop()** method is called at least once in an applet's life, when the browser leaves the page in which the applet is embedded. The applet's start() method will be called if at some later point the browser returns to the page containing the applet. In some cases the stop() method may be called multiple times in an applet's life.. Your applet should use the stop() method to pause any running threads.

The **destroy()** method is called exactly once in an applet's life, just before the browser unloads the applet. This method is generally used to perform any final clean up. For example, an applet that stores state on the server might send some data back to the server before it's terminated.

12.7 Creating executable Applet

To create an executable applet, compile the applet .java extension file using javac.exe file. For ex: javac myapplet.java. The result of compilation generates myapplet.class file.

12.8 Designing a Web Page HTML file

In fact, there are two ways in which you can run an applet:

■ Executing the applet within a Java-compatible Web browser using HTML file.
■ Using an applet viewer, such as the standard SDK tool, **appletviewer**.

Using web browser to view Applet

Applet Tag

In order to view the applet in a Web browser, you need to write a short HTML text file. This code contains the APPLET tag. For example,

TestApplet :

```
<html>
<applet code="TestApplet" width=200 height=60>
```

```
</applet>
</html>
```

The **width** and **height** statements specify the dimensions of the display area usedby the applet. After you create this file, you can execute your browser and then load this file, which causes **TestApplet** to be executed.

12.9 Running the applet

We can run the applet either using html file or appletviewer.

To execute **TestApplet** with an applet viewer, you may also execute the HTMLFor example, if the preceding HTML file is called **RunApp.html**, then the following command line will run **TestApplet**:

```
C:\>appletviewer RunApp.html
```

However, a more convenient method exists that you can use to speed up testing.

Note :

You can also include a comment at the head of your Java source code file that contains theAPPLET tag. If you use this method, the **TestApplet** source file looks like this:

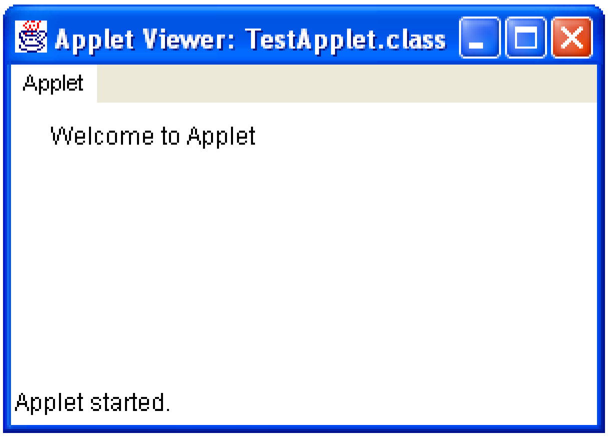
```
import java.awt.*;
import java.applet.*;

/* <applet code="TestApplet" width=200 height=60>
</applet> */
public class TestApplet extends Applet
{
public void paint(Graphics g)
{
        g.drawString("Welcome to Applet", 20, 20);

}
}
```

Now, Execute the applet viewer, specifying the name of your applet's source file. Theapplet viewer will encounter the APPLET tag within the comment and executeyour applet.

Output :



12.10Passing Parameters to Applets

Parameters are passed to applets in <PARAM> tags between the opening and closing APPLET tags. Inside the applet, you read the values passed through the PARAM tags with the getParameter() method of the java.applet.Applet class. For example

```
import java.applet.*;
import java.awt.*;
 public class ParameterEx extends Applet {
  private String strDefault = "Myfirst Java Applet.";
  public void paint(Graphics g) {
    String str = this.getParameter("Message");
    if (str == null)     str = strDefault;
    g.drawString(str, 50, 25);   }
}
```

HTML CODE :

```
<HTML>
<TITLE>Passing Parameter in Java Applet</TITLE>
<BODY>
This is the My applet:<P>
<APPLET code="ParameterEx.class" width="500" height="200">
<PARAM name="message" value="Passing parameter example.">
</APPLET>
</BODY>
</HTML>
```

To run the program using appletviewer, go to command prompt and type *appletviewer appletParameter.html* Appletviewer will run the applet for you.

12.11 Displaying Numerical values

Here is an example program to display numbers from 1 to 100 using applet.

```
/*  <applet code=Print1To100Num.class height=300 width=250>
</applet>*/
import java.awt.*;  // import awt package
import java.applet.*; // import applet package
public class Print1To100Num extends Applet
{
   public void paint(Graphics g)
   {
      for(int i=1;i<=100;i++)
      {
g.drawString(i+"",20,10*i); // display message on applet
      }
   }
}
```

12.12 Getting input from the user

Applets work in a graphical environment . Therefore , applets treat inputs as text strings .

```
import java.awt.*;
import java.applet.*;
public class GettingInputfromtheUserEx extends Applet
{
         TextField t1, t2;
         public void init()
      {
            t1 = new TextField(10);
            t2 = new TextField(10);
            add(t1);
            add(t2);
            t1.setText("0");
            t2.setText("0");
```

NOTES

```
        }
        public void paint(Graphics g)
        {
            int a=0,b=0,c=0;
            String str1,str2,str;
            g.drawString("Enter the number in each box",10,50);
            try
            {
                str1=t1.getText();
                a=Integer.parseInt(str1);
                str2=t2.getText();
                b=Integer.parseInt(str2);
            }
            catch(Exception e){     }
            c=a+b;
            str=String.valueOf(c);
            g.drawString("a =",10,15);
            g.drawString(a,20,75);
            g.drawString("b =",30,15);
            g.drawString(b,40,75);
          g.drawString("Sum is",50,15);
            g.drawString(str,100,75);
        }
}
```

HTML CODE TO RUN APPLET

```
<HTML>
<HEAD>
        <TITLE>Getting Input from the User</TITLE>
</HEAD>
<BODY>
        <APPLET Code="GettingInputfromtheUserEx.class" Width=400
Height=300>
        </APPLET>
</BODY>
</HTML>
```

12.13 Check Your Progress Questions

1 What is an applet?
2 What does the <applet> tag specify?

12.14 Answers to Check Your Progress Questions

1. Applet is a small program typically embedded with in a web page to create dynamic and interactive web applications and runs under any java enabled browser.

2. The <applet> tag is used to specify the location of the executable class file which invoke an applet.

12.15 Summary

Applet is a small program typically embedded with in a web page to create dynamic and interactive web applications and runs under any java enabled browser.In java, java.applet.Applet class has four methods defines the lifecycle of the servlets, They are,public void init();public void start();public void stop();public void destroy();

12.16 Key Words

- **init()** method is called exactly once in an applet's life, when the applet is first loaded. It's normally used to read PARAM tags, start downloading any other images or media files you need, and set up the user interface.

- **start()** method is called at least once in an applet's life, when the applet is started or restarted. A start() method is often used to start any threads the applet will need while it runs.

- **stop()** method is called at least once in an applet's life, when the browser leaves the page in which the applet is embedded. The applet's start() method will be called if at some later point the browser returns to the page containing the applet.

- **destroy()** method is called exactly once in an applet's life, just before the browser unloads the applet. This method is generally used to perform any final clean up.

NOTES

12.17 Self-Assessment Questions and Exercises

1) Define Applet.
2) What are the limitations of Applet?
3) Write a simple program to illustrate Applet program.
4) How to define and invoke applets? Explain.
5) Discuss the life cycle of an Applet.
6) Briefly explain how parameters are passed to applet.

12.18Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998

NOTES

BLOCK 5
MANAGING INPUT/OUTPUT FILES IN JAVA

UNIT 13
INTRODUCTION TO INPUT / OUTPUT

Structure

13.1 Introduction

The Java Input/Output (I/O) is a part of **java.io** package. The java.io package contains a relatively large number of classes that support input and output operations. Java does provide strong, flexible support for I/O as it relates to files and networks.

The classes in the java.io package are primarily abstract classes and stream-oriented that define methods and subclasses which allow bytes to be read from and written to files or other input and output sources. All data in java is written and read using streams.

13.2 Objectives

After going through the unit you will be able to;

- Understand I/O streams

NOTES

- Learn about various stream classes
- Write program using Byte stream classes
- Able to understand about character stream classes

13.3Stream &Stream classes

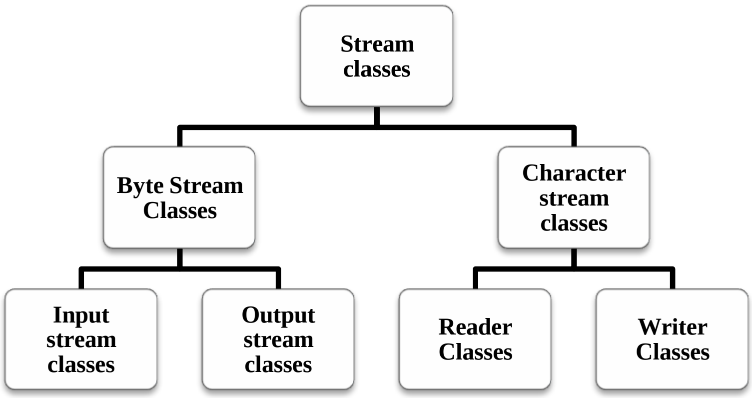
Stream

Java programs perform I/O through streams. A stream is an abstraction that either produces or consumes information. A stream is a path traveled by data in a program A stream is linked to a physical device by the Java I/O system. An input stream can abstract many different kinds of input: from a disk file, a keyboard, or a network socket. Likewise, an output stream may refer to the console, a disk file, or a network connection. Java implements streams within class hierarchies defined in the java.io package.

Stream Classes

Java defines two types of streams:

- Byte streams
- Character streams.



13.1 Classification of stream classes

B*yte streams* provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary data.*Character streams* provide a convenient means for handling input and output of characters.

These streams are declared in the java.lang.System class and they are all byte streams. For example, using some of its methods, you can obtain the current time and the settings of various properties associated with the system. **Systems also contain three predefined stream variables in, out, and err**. These fields are declared as **public** and **static** within **System**.

This means that they can be used by any other part of your program and without reference to a specific **System** object.

- **System.out** refers to the standard output stream. By default, this is the console.
- **System.in** refers to standard input, which is the keyboard by default.
- **System.err** refers to the standard error stream, which also is the console by default.

Note: **System.in** is an object of type **InputStream.**
System.out and **System.err** are objects of type **PrintStream**.

For example Accepting input through keyboard **Using standard Streams :**

```
import java.io.*;
class stdin{
        Public static void main(String args[]){
                byte name = new byte[10];
                System.out.println("What is your name?");
                try{
                        System.in.read(name);
                        System.out.write("Hello "+name);
                    }catch(IOException e){
                        System.out.println("Error");}      }
}
```

13.3.1 Byte stream classes

To use the stream classes, you must import **java.io**.Byte streams are defined by using two class hierarchies. At the top are two abstract classes: **InputStream** and **OutputStream**. Each of these abstract classes has several concrete subclasses.

The abstract classes **InputStream** and **OutputStream** define several key methods that the other stream classes implement. Two of the most important are **read( )** and **write( )**, which, respectively, read and write bytes of data.

Input Steam

The **InputStream** class is used for reading the data such as a byte and array of bytes from an input source. An input source can be a **file**, a **string**, or **memory** that may contain the data

An input stream is automatically opened when you create it. You cans explicitly close a stream with the **close( )** method, or let it be closed implicitly when the object is found as a garbage.

OutputStream

The **OutputStream class** is a sibling to InputStream that is used for writing byte and array of bytes to an output source. Similar to input sources, an output source can be anything such as a file, a string, or memory containing the data.

You can explicitly close an output stream with the **close( )** method, or let it be closed implicitly when the object is garbage collected. The following listings of classes are provided by the **java.io** package for byte stream classes shown in the table 13.1:

Table 13.1 Byte stream classes

Class	**Description**
BufferedInputStream	contains methods to read bytes from the buffer (memory area)
ByteArrayInputStream	contains methods to read bytes from a byte array
DataInputStream	contains methods to read Java primitive data types
FileInputStream	contains methods to read bytes from a file
FilterInputStream	contains methods to read bytes from other input streams which it uses as its basic source of data
ObjectInputStream	contains methods to read objects
PipedInputStream	contains methods to read from a piped output stream. A piped input stream must be connected to a piped output stream
SequenceInputStream	contains methods to concatenate multiple input streams and then read from the combined stream

13.3.2 Character Stream classes

Reader and **Writer** are the abstract super classes for character stream sin java.io package. The abstract classes **Reader** and **Writer** define several key methods that the other stream classes implement. Two of the most important methods are **read( )** and **write( )**, which read and write characters of data, respectively. The following listings of classes are provided by the **java.io** package for character stream classes shown in the table 13.2:

Table 13.2 Character Stream classes

Class	Description
BufferedReader	contains methods to read characters from the buffer
CharArrayReader	contains methods to read characters from a character array
FileReader	contains methods to read from a file
FilterReader	contains methods to read from underlying character-input stream
InputStreamReader	contains methods to convert bytes to characters
PipedReader	contains methods to read from the connected piped output stream
StringReader	contains methods to read from a string

The PrintWriter Class

The recommended method of writing to the console when using Java is through a **PrintWriter** stream. **PrintWriter** is one of the character-based classes. Using a character-based class for console output makes it easier to internationalize your program. **PrintWriter** defines several constructors.

PrintWriter(OutputStream *outputStream*, boolean *flushOnNewline*)

Here, *outputStream* is an object of type **OutputStream**, and *flushOnNewline* controls
whether Java flushes the output stream every time a **println( )** method is called. If*flushOnNewline* is **true**, flushing automatically takes place. If **false**, flushing is notautomatic.

PrintWriter supports the **print( )** and **println( )** methods for all types including**Object**. If an argument is not a simple type, the **PrintWriter** methods call theobject's **toString( )** method and then print the result.

To write to the console by using a **PrintWriter**, specify **System.out** for the outputstream and flush the stream after each newline. For example, this line of code creates a **PrintWriter** that is connected to console output

```
PrintWriter pw = new PrintWriter(System.out, true);
```

The following application illustrates using a **PrintWriter** to handle console output:

```
// Example PrintWriter

import java.io.*;

public class PrintWriterExample
```

```
{
public static void main(String args[])
{
PrintWriter pw = new PrintWriter(System.out, true);
pw.println("Hello Java world");
int i = 1000;
pw.println(i);
double d = 45.0;
pw.println(d);
}
}
```

Output:

```
Hello Java world
1000
45.0
```

Character Streams for Files

Character streams are used to work with any text using files The FileReader and FileWriter classes are used for the text manipulation.

FileReader class

The **FileReader** class creates a **Reader** that you can use to read the contents of a file. Its two most commonly used constructors are shown here:

FileReader(String *filePath*)
FileReader(File *fileObj*)

Either can throw a **FileNotFoundException**. Here, *filePath* is the full path name of a file, and *fileObj* is a **File** object that describes the file.

The following example shows how to read lines from a file and print these to the standard output stream. It reads its own source file, which must be in the current directory.

```
// Example FileReader.

import java.io.*;

class FileReaderDemoPro
{
public static void main(String args[]) throws Exception
{
        FileReader fr = new FileReader("HELLO.java");
        BufferedReader br = new BufferedReader(fr);
```

NOTES

```
        String s1;
        while((s1 = br.readLine()) != null)      {
        System.out.println(s1);
        }
  fr.close();
  }
 }
```

FileWriter class

FileWriter creates a **Writer** that you can use to write to a file. Its most commonly used constructors are shown here:

FileWriter(String *filePath*)
FileWriter(String *filePath,* boolean *append)*
FileWriter(File *fileObj*)

They can throw an **IOException** or a **SecurityException**. Here, *filePath* is the full path name of a file, and *fileObj* is a **File** object that describes the file. If *append* is **true**, then output is appended to the end of the file.

Creation of a **FileWriter** is not dependent on the file already existing. **FileWriter** will create the file before opening it for output when you create the object. In the case where you attempt to open a read-only file, an **IOException** will be thrown.

For Example the following program will copy a file from one to other.

```
// Example FileWriter.

import java.io.*;

class FileWriterDemotocopyfile
{
public static void main(String args[]) throws Exception
{
        File inputfile= new File("Hello.java");
        File outputfile = new File("Hello1.java");
        FileReader fr = new FileReader(inputfile);
        FileWriter fw = new FileWriter(outputfile);
        int c;
        while((c=fr.read()) != -1)
        fw.write(c);
        fr.close();
        fw.close();
}
}
```

13.4 Check Your Progress Questions

1. What are the types of character classes?
2. Name the two class hierarchies of byte streams.

13.5 Answers to Check Your Progress Questions

1. Reader and Writer are the abstract super classes for character stream sin java.io package.
2. Byte streams are defined by using two class hierarchies. At the top are two abstract classes: InputStream and OutputStream.

13.6 Summary

A stream is a path traveled by data in a program A stream is linked to a physical device by the Java I/O system.Java defines two types of streams namely Byte streams and Character streams.

13.7 Key Words

- **Stream** A stream is an abstraction that either produces or consumes information.
- **System.out** refers to the standard output stream. By default, this is the console.
- **System.in** refers to standard input, which is the keyboard by default.
- **System.err** refers to the standard error stream, which also is the console by default.

13.8 Self-Assessment Questions and Exercises

1) What are streams?
2) What are predefined streams supported in java?
3) What are character streams? Explain about various character stream classes.
4) What are Byte streams? Explain about byte stream classes with examples.

NOTES

13.9 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.
4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998.

UNIT 14
INPUT / OUTPUT (I/O) CLASSES

Structure

14.1 Introduction

Most of the real-life applications require a large amount of input and output data, which is difficult to manage using the commonly used console Input/Output (I/O) devices like keyboard and screen. Java supports input and output of data through the classes included in the java.io package. This unit will take you through the various aspects of input/output handling in Java.

14.2 Objectives

After going through the unit you will be able to;

- Know using stream &the file class
- Learn about various Input / output Exceptions
- Understand how to create files
- Write programs for Reading / writing characters
- Write programs for Reading writing bytes
- Understand about Random access files
- Learn about Interactive input and output

14.3 Using Stream

A stream is an abstraction that either produces or consumes information. The following listings of classes are provided by the **java.io** package tabulated in Table 13.3

Table 13.3 java.io.package classes

Class	Description
BufferedInputStream	It used for creating an internal buffer array. It supports the mark and reset methods.
BufferedOutputStream	This class used for writes byte to output stream. It implements a buffered output stream.
BufferedReader	This class provides read text from character input stream and buffering characters. It also reads characters, arrays and lines.
BufferedWriter	This class provides write text from character output stream and buffering characters. It also writes characters, arrays and lines.
ByteArrayInputStream	It contains the internal buffer and read data from the stream.
ByteArrayOutputStream	This class used for data is written into byte array. This is to implement in output stream class.
CharArrayReader	It used for char input stream and implements a character buffer.
CharArrayWriter	This class also implements a character buffer and it uses a writer.
DataInputStream	This class reads the primitive data types from the input stream in a machine format.
DataOutputStream	This class writes the primitive data types from the output stream in machine format.
File	This class shows a file and directory pathnames.
FileDescriptor	This class uses for create a FileInputStream and

	FileOutputStream.
FileInputStream	It contains the input byte from a file and implements an input stream.
FileOutputStream	It uses for writing data to a file and also implements an output stream.
FilePermission	It provides the permission to access a file or directory.
FileReader	This class used for reading characters file.
FileWriter	This class used for writing characters files.
FilterInputStream	This class overrides all methods of InputStream and contains some other input stream.
FilterOutputStream	This class overrides all methods of OutputStream and contains some other output stream.
FilterReader	It reads the data from the filtered character stream.
FilterWriter	It writes data from the filtered character stream.
InputStream	This class represents an input stream of bytes.
InputStreamReader	It reads bytes and decodes them into characters.
LineNumberReader	This class has a line numbers
ObjectInputStream	This class used for recover the object to serialize previously.
ObjectInputStream.GetField	This class access to president fields read form input stream.
ObjectOutputStream	This class used for write the primitive data types and also writes the object to read by the ObjectInputStream.
ObjectOutputStream.GetField	This class access to president fields write in to ObjectOutput.
ObjectStreamClass	Serialization's descriptor for classes.
ObjectStreamField	This class describes the serializable field.
OutputStream	This class represents an output stream of bytes.
OutputStreamWriter	It writes bytes and decodes them into characters.
PipedInputStream	In this class the data bytes are written into piped output stream.

	This class also connected into a piped output stream.
PipedOutputStream	This class also communicates the piped input stream into piped output stream. It creates communication between both.
PipedReader	It is a piped character-input stream.
PipedWriter	It is a piped character-output stream.
PrintStream	This class adds the functionality of another output stream.
PrintWriter	This class adds the functionality of another input stream.
PushbackInputStream	It also includes the function of input stream. Such as: "push back" or "unread" one byte.
PushbackReader	This is a character stream reader and reads the data push back into the stream.
RandomAccessFile	It supports both reading and writing to a random access file.
Reader	It used for reading character stream.
SequenceInputStream	It represents the logical concatenation of other input stream.
SerializablePermission	This is a serializable permission class.
StreamTokenizer	It takes an input stream and parses it into "tokens" . The token to be allowed at the read time.
StringReader	This is a character string class. It has character read source.
StringWriter	This is also a character string class. It uses to shows the output in the buffer.
Writer	It uses for writing to character stream.

14.4 Using 'File' class

The File class allows you to obtain and manipulate the information about a file such as permissions, size, time and so on. Unlike the other classes of java.io package, this class does not operate on the streams; it deals directly

NOTES

with the files and file system. That is, it does not specify how the data is retrieved from or sent to the files. Using this class, you can also make new directories, rename as well as delete the files.

A object can be created using any one of the following constructors;

- File(File parent, String child) : Creates a new File instance from a parent abstract pathname and a child pathname string.
- File(String pathname) : Creates a new File instance by converting the given pathname string into an abstract pathname.
- File(String parent, String child) : Creates a new File instance from a parent pathname string and a child pathname string.
- File(URI uri) : Creates a new File instance by converting the given file: URI into an abstract pathname.

The following program illustrates the file class to display files property;

```
class fileProperty
{
   public static void main(String[] args)
{
      //accept file name or directory name through command line args
      String fname =args[0];

      //pass the filename or directory name to File object
      File f = new File(fname);

      //apply File class methods on File object

      System.out.println("File name :"+f.getName());
      System.out.println("Path: "+f.getPath());
      System.out.println("Absolute path:" +f.getAbsolutePath());
      System.out.println("Parent:"+f.getParent());
      System.out.println("Exists :"+f.exists());

      if(f.exists())
      {
         System.out.println("Is writeable:"+f.canWrite());
         System.out.println("Is readable"+f.canRead());
         System.out.println("Is a directory:"+f.isDirectory());
         System.out.println("File Size in bytes "+f.length());
      }
   }
}
```

Output

```
File name :file.txt
Path: file.txt
Absolute path:C:\Users\akki\IdeaProjects\codewriting\src\file.txt
Parent:null
Exists :true
Is writeable:true
Is readabletrue
Is a directory:false
File Size in bytes 20
```

14.5 Input / Output Exceptions

I/O exception occurs when an IO operation has failed for some reason. It is also a checked exception which means that your program has to handle it.IOException has many sub classes that are specific in nature. That means, when your application searching to read a file, if the file is not found that there is a FileNotFoundException to be thrown. FileNotFoundException is a subclass of IOException.For example;(FileNotFoundException)

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
/**
 * File Not Found Exception example
 * @author Krishna
 *
 */
public class JavaFileExample
{
        public static void main(String[] args)
{
                File file = new File("D:/JavaTest.txt");
                FileInputStream fileInputStream = null;
                try
{
                        fileInputStream = new FileInputStream(file);
                        while (fileInputStream.read()!=-1){
                                System.out.println(fileInputStream.read());
                        }
                }catch (FileNotFoundException e){
                        e.printStackTrace();
                }catch (IOException e){
```

```
                        e.printStackTrace();
                }finally{
                        try{
                                fileInputStream.close();
                        }catch (IOException e)
{
                                e.printStackTrace();
                        }
                }
        }
}
```

The above program would throw an exception if the file "JavaText.txt" is not in the mentioned path. You will get the following exception.

The java.io.Exceptions provides for system input and output through data streams, serialization and the file system.

Data Streams

Data streams perform binary I/O operation on primitive data type values (boolean, char, byte, short, int, long, etc.) as well as on String values. If you need to work with data that is not represented as bytes or characters then you can use Data Streams. These streams filter an existing byte stream so that each primitive data types can be read from or written to the stream directly. The two data streams are DataInputStream and DataOutputStream.

The methods to read and write data inputs are,

Read	**Write**
readBoolean()	writeBoolean()
readByte()	writeByte()
readDouble()	writeDouble()
ReadFloat()	writeFloat()
ReadLong()	writeLong()
ReadShort()	writeShort()
readInt()	writeInt()

These input methods returns the primitive data type indicated by the name of the method.

DataInputStream

A data input stream is created with the constructor of **DataInputStream** class. The constructor of DataInputStream is written as:

```
DataInputStream(java.io.InputStream in);
```

The **read( )** method is used to read the data according to its types. For example, the **readInt( )** method reads the **int** type of value while the **readFloat()** method reads the fraction value. The **readLine( )** Methods reads the string per line from a file. for Example,

```
import java.io.*;
public class DataIstr{
pubic static void main(String args[]){
try{
        FileInputStream file = new FileInputStream("e.dat")
        DataInputStream data = new DataInputStream(file);
        try{
         while{true){
                int in = data.readInt();
                    System.out.print(in+"  ");
        }
        }catch(EOFException eof){data.close();}
        }catch(IOException e){
        System.out.println("Error "e.toString());       }
}}
```

DataOutputStream

It writes only Java primitive data types and doesn't write the object values. A data output stream is created with the constructor of **DataOutputStream** class. The constructor of DataOutputStream is written as:

```
DataOutputStream(java.io.OutputSrream out);
```

The **write( )** method is used to write the data according to its types. For example, the **writeInt( )** method writes the **int** type of value while the **writeFloat()** method writes the fraction value. The **writeUTF( )** method writes the string per line to a file. for example,

```
import java.io.*;
public class DataOstr{
pubic static void main(String args[]){
try{
        FileInputStream file = new FileInputStream("e.dat")
        DataInputStream data = new DataInputStream(file);
        for(int i =0 ; i<50;i++)
        data.writeInt(i*2);
        data.close();
  }catch(IOException e)
{
        System.out.println("Error "+.toString());       }
```

NOTES

```
        }
        }
```

14.6 Creation of files

Java provides a number of classes and methods that allow you to read and write (create) files. In Java, all files are byte-oriented, and Java provides methods to read and write bytes from and to a file. However, Java allows you to wrap a byte-oriented file stream within a character-based object.

File Output Stream

A file Output stream can be created with the FileOutputStream(String) constructor.

FileOutputStream(String *fileName*) throws FileNotFoundException

The String argument should be the name of the string. For output streams, if the file cannot be created, then **FileNotFoundException** is thrown. To write to a file, you will use the **write( )** method defined by **FileOutputStream**. Its simplest form is shown here:

void write(int *byteval*) throws IOException

This method writes the byte specified by *byteval* to the file. Although *byteval* is declared as an integer, only the low-order eight bits are written to the file. If an error occurs during writing, an **IOException** is thrown.

When you are done with a file, you should close it by calling **close( )**.

void close( ) throws IOException

The following program uses write**( )** method to write to file .

```
import java.io.*;

public class WriteBytesDemo
{
        public static void main(String args[])
        {
        int data[]={1,2,3,4,5,6,7,8,9,10};
        try{
            FileOutputStream file = new FileOutputStream("outfile");
                for(int i=0;i<data.length;i++)
                 file.write(data[i]);
                 file.close();
          }catch(IOException e){
                System.out.println("I/O Error ");
```

```
        }
          }
        }
```

The following program uses write**( )** method to write text to text file .

```
/* Write a text to text file.

import java.io.*;
public class WriteFile
{
   public static void main(String[] args) throws IOException
{
     File f=new File("writefile.txt");
     FileOutputStream fop=new FileOutputStream(f);
     if(f.exists())
    {
        String str="This data is written through the program";
        fop.write(str.getBytes());
        fop.flush();
        fop.close();
        System.out.println("The data has been written");
        }
        else
          System.out.println("This file is not exist");
   }
  }
```

File Input Streams

A file input stream can be created with FileInputStream Constructor.

FileInputStream(String *fileName*) throws FileNotFoundException

The String argument should be the name of the string. When you create an input stream, if the file does not exist, then **FileNotFoundException** is thrown. After you create a file input stream, you can read bytes from the stream by calling its read() method.

int read( ) throws IOException

The method returns an integer contain the next byte in the stream. If the method returns -1 it identifies that the end of the file stream has been reached. It can throw an **IOException**. When you are done with a file, you should close it by calling **close( )**.

NOTES

void close( ) throws IOException

The following program uses **read( )** to input and display the contents of a file.

```
import java.io.*;
public class ReadBytesDemo{
public static void main(String args[]){
try{
 FileInputStream file = new FileInputStream("Hello.class");
        boolean eof= false;
        int c=0;
        while(!eof){
          int inp = file.read();
          System.out.println(input+" ");
             if(inp == -1) eof = true;
          else c++;
             }
         file.close();
            System.out.println("\n Output Bytes read :" "+c);
          }catch(IOException e){
            System.out.println("Error -- "+e.toString());
}
}
}

/* Read a text file. */

import java.io.*;

public class ReadFile{
  public static void main(String[] args) throws IOException{
    File f;
   f=new File("myfile.txt");

    if(!f.exists()&& f.length()<0)
    System.out.println("The specified file is not exist");

    else{
      FileInputStream finp=new FileInputStream(f);
    byte b;
   do{
    b=(byte)finp.read();
    System.out.print((char)b);
   }
    while(b!=-1);
      finp.close();
      }
```

```
    }
  }
```

Output:

My Text file contents are displayed here.

14.7 Reading / writing characters

Reading Characters

To read a character from a **BufferedReader**, use **read( )**. The version of **read( )** that you will be using is int read( ) throws IOException. Each time that **read( )** is called, it reads a character from the input stream and returns it as an integer value. It returns –1 when the end of the stream is encountered. The following program demonstrates **read( )** by reading characters from the console until the user types a "q":

```
// Use a BufferedReader to read characters from the console.

import java.io.*;

class BuffReader
{
public static void main(String args[]) throws IOException{
char c;
BufferedReader br = new
BufferedReader(new InputStreamReader(System.in));
System.out.println("Enter characters, 'q' to quit.");
do {
c = (char) br.read();
System.out.println(c);
 } while(c != 'q');
 }
}
```

Output:

```
Enter characters, 'q' to quit.
HELLOq

H
E
L
L
O
```

Reading Strings

To read a string from the keyboard, use the version of **readLine( )** that is a member ofthe **BufferedReader** class. Its general form is shown here:

String readLine( ) throws IOException

The following program reads and displays lines of text:

// Read a string from console using a BufferedReader.

```
import java.io.*;
 public class ReadStandardIOstring
{
  public static void main(String[] args) throws IOException
{
      InputStreamReader inp = new InputStreamReader(System.in)
      BufferedReader br = new BufferedReader(inp);
      System.out.println("Enter text : ");
     String str = in.readLine();
     System.out.println("You entered String : ");
     System.out.println(str);
  }
}
```

14.8 Reading / writing bytes

Reading bytes

The basic class in java.io to read data is java.io.InputStream. All classes that read bytes are derived from this class. This class is extended by the following classes.

FileInputStream

This stream reads raw bytes from a file. The read methods in this class return a byte of data read from a file.

ObjectInputStream

This class is used to read objects written using ObjectOutputStream. It deserializes the primitive data types and objects

PipedInputStream

A unix ‘pipe’ like implementation can be accomplished using this class. It can connect to a pipedoutputsteam and read data off it.

BufferedInputStream

This is probably the most used class. It buffers the data from any of the above inputstream. The methods in this class increase reading efficiency since they try to read the maximum number of bytes that can be read from the file system in one operation.

ByteArrayInputStream

This class contains a buffer(array) or bytes that store the next few bytes that will be read from the stream.

Example :

```
import java.io.FileInputStream;
import java.io.InputStream;
public class InputStreamDemo {
  public static void main(String[] args) throws Exception {
    InputStream is = null;
    byte[] buffer = new byte[5];
    char c;
        try {
      // new input stream created
      is = new FileInputStream("C://testread.txt");

      System.out.println("Characters printed:");

      // read stream data into buffer
      is.read(buffer);

      // for each byte in the buffer
      for(byte b:buffer) {

        // convert byte to character
        c = (char)b;

        // prints character
        System.out.print(c);
      }

    } catch(Exception e) {
      // if any I/O error occurs
      e.printStackTrace();
    } finally {
      // releases system resources associated with this stream
```

```
        if(is!=null)
          is.close();
      }
    }
}
```

Writing bytes

All classes that write bytes extend java.io.OutputStream. The important classes are:

FileOutputStream

The methods in this class write bytes of data to a file. Note that this writes raw bytes and not characters.

ObjectOutputStream

It writes primitive java types and objects to an ouputstream. The data could be written to a file or to a socket. Data written using this method can be read back using the ObjectInputStream

PipedOutputStream

A piped output stream connects to aPipedInputStream to read bytes. Data may be written by one thread and read by another.

BufferedOutputStream

It buffers data from an input stream and writes the buffered data. It is an efficient method of writing data since the operating systems may write an array of bytes in a single operation and invoking write operation for each byte may be inefficient

PrintStream

It wraps an InputStream and adds functionality to print various representations of data values. It never throws IOException and there is an option for automatic flushing

ByteArrayOutputStream-This class writes bytes at an array of bytes.

14.9Using RandomAccessFile class

The RandomAccesFile class is used for reading and writing to random access file. A random access file behaves like a large array of bytes. There

is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations. The constructors for the RandomAccessFile class are;

RandomAccessFile(File file, String mode)

Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.

RandomAccessFile(String name, String mode)

Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Java provides java.io.RandomAccessFile class that enables you to perform random access file input and output operations as opposed to sequential file I/O offered by ByteStream and CharacterStream classes.

When a data file is opened for random read and write access, an internal file pointer is set at the beginning of the file. When you read or write data to the file, the file pointer moves forward to the next data item. For example, when reading an in t value using readlnt() , 4 bytes are read from the file and the file pointer moves 4 bytes ahead from the previous file pointer position.

Similarly, when reading a double value using readDouble () , 8 byte are read from the file pointer and the file pointer moves 8 bytes ahead from the previous file pointer position.

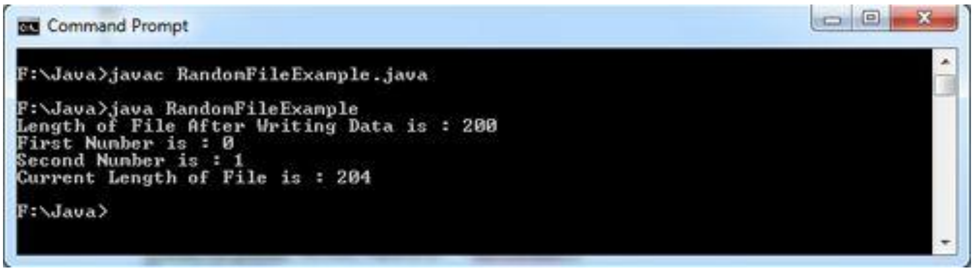
```
import java.io.*;

class RandomFileExample
{
    public static void main(String[] args)
   {
        try
        {
           RandomAccessFile file = new RandomAccessFile("std.dat","rw");
           file.setLength(0);
           for(int i=0;i<50;i++)
               file.writeInt(i);
               System.out.println("Length of File After Writing Data is :
"+file.length());
               file.seek(0);
               System.out.println("First Number is : "+file.readInt());
               file.seek(1*4);
               System.out.println("Second Number is : "+file.readInt());
               file.writeInt(101);
```

NOTES

```
                file.seek(file.length());
                file.writeInt(50);
                System.out.println("Current Length of File is : "+file.length());
}
                catch(Exception e)
               {
                    e.printStackTrace();
               }
   }
}
```

Output



14.10 Interactive input and output

A standard task in Java programming is to get interactive input from the user; that is, toread in a number or a string typed at the keyboard.

Reading Console Input

In Java, console input is accomplished by reading from **System.in**. To obtain a character-based stream that is attached to the console, you wrap **System.in** in a **BufferedReader** object, to create a character stream. **BuffereredReader** supports a buffered input stream. BufferedReader Constructor,

BufferedReader(Reader *inputReader*)

Here, *inputReader* is the stream that is linked to the instance of **BufferedReader** that is being created. **Reader** is an abstract class. One of its concrete subclasses is **InputStreamReader**, which converts bytes to characters. To obtain an **InputStreamReader** object that is linked to **System.in**, use the following constructor:

InputStreamReader(InputStream *inputStream*)

Because **System.in** refers to an object of type **InputStream**, it can be used for *inputStream*. Putting it all together, the following line of code creates a **BufferedReader** that is connected to the keyboard:

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

where br is a character-based stream that is linked to the console through **System.in**.

Writing Console Output

PrintStream is an output stream derived from **OutputStream**, it also implements the low-level method **write( )**. Thus, **write( )** can be used to write to the console.

The simplest form of **write( )** defined by **PrintStream** is shown here:

void write(int *byteval*)

This method writes to the stream the byte specified by byteval. Although byteval is declared as an integer, only the low-order eight bits are written.

Example that uses **write( )** to output the character “J” followed by a newline to the console:

```
class PrintDemo
{
public static void main(String args[])
{
int b; b = 'J';
System.out.write(b);
System.out.write('\n');
}
}
```

14.11 Check Your Progress Questions

1. When I/O exception occurs?

2. What is the purpose of Data Streams?

14.12 Answers to Check Your Progress Questions

1. I/O exception occurs when an IO operation has failed for some reason. It is also a checked exception which means that your program has to handle it.

NOTES

2. Data streams perform binary I/O operation on primitive data type values (boolean, char, byte, short, int, long, etc.) as well as on String values.

14.13 Summary

A stream is an abstraction that either produces or consumes information.The File class allows you to obtain and manipulate the information about a file such as permissions, size, time and so on. Unlike the other classes of java.io package, this class does not operate on the streams; it deals directly with the files and file system.Java provides a number of classes and methods that allow you to read and write (create) files. In Java, all files are byte-oriented, and Java provides methods to read and write bytes from and to a file.The RandomAccesFile class is used for reading and writing to random access file. A random access file behaves like a large array of bytes.

14.14 Key Words

- **Stream**is an abstraction that either produces or consumes information.

- **RandomAccesFile** class is used for reading and writing to random access file. A random access file behaves like a large array of bytes.

14.15 Self-Assessment Questions and Exercises

1) Write short note on PrintWriter class.
2) Briefly explain about File streams.
3) What is the use of Data streams? Explain about DataStream classes.
4) Explain in detail about RandomAccessFile class? What are its constructors and methods?
5) Discuss in detail about interactive input / output operations in Java.

14.16 Further Readings

1. R. Krishnamoorthy and S. Prabhu, Internet and Java Programming, New Age International Publishers, 2004
2. Programming with Java, 4e, E. Balagurusamy, Tata McGraw-Hill, 2010.
3. Deitel, Deitel and Nieto, Internet and World Wide Web – How to program, Pearson Education, 2000.

4. Naughton and H.Schildt, Java 2 - The complete reference, Tata McGraw-Hill, Fourth edition, 2006.
5. Elliotte Rusty Harold, Java Network Programming, O'Reilly Publishers, 2000.
6. B.Mohamal Ibrahim , Java : J2SE – A Practical Approach, Firewall media, 2006.
7. Cay S. Horstmann, Gary Cornell, Core Java, Volume I and II, 5th Edition, Pearson Education, 2003.
8. Topley, J2ME in A Nutshell, O'Reilly Publishers, 2002.
9. Hunt, Guide to J2EE Enterprise Java, Springer Publications, 2004.
10. Ed Roman, Enterprise Java Beans, Wiley Publishers, 1998

*****

MODEL QUESTION PAPER

DISTANCE EDUCATION

B.SC (INFORMATION TECHNOLOGY) EXAMINATION

INTERNET AND JAVA PROGRAMMING
Second Year - Third Semester
(CBCS – 2018-19 Academic Year Onwards)

Time : 3 hours | Max Marks :75

PART - A (10 x 2=20 Marks)
Answer all questions.

1. List the names of Internet Service Providers (ISP) in India.
2. Name any two search engines.
3. What is meant by virtual machine?
4. Define Data abstraction.
5. Write short note on : Java Support System
6. What is the use of 'final' keyword?
7. Define Applet.
8. What are the types of errors?
9. What is the purpose of data stream classes?
10. State the importance of synchronization.

PART - B (5 x 5 Marks = 25 Marks)
Answer all questions choosing either (a) or (b)
.

11. (a)Briefly explain about Domain Name system.
Or
(b) Write short note on : Connecting internet.

12. (a) How java differs from C++? Explain.
Or
(b)Briefly explain about the applications of Java.

13. (a) Summarize the use of Wrapper classes and its methods.
Or
(b) Write a java program to illustrate method overriding.

14. (a) Write an applet program to draw a car.
Or
(b) Discuss about the life cycle of a thread with neat sketch.

15. (a) Explain about handling Random Access files in Java.
Or
(b) Write short note on : Input / Output Exception

Part – C (3 x 10 = 30 Marks)

Answer any three questions.

16. Discuss in detail about e-mail communication with its features
17. Elucidate about any two types of inheritances with neat diagram and example.
18. How will you handle multiple catch clauses? Explain with example.
19. Describe about basic concepts of object oriented programming? What are its advantages? Explain.
20. Explain in detail about reading / writing bytes in Java. Give Example.

****

Master of Computer Applications
31543

INTERNET AND JAVA PROGRAMMING

IV - Semester



ALAGAPPA UNIVERSITY

[Accredited with'A+' Grade by NAAC (CGPA:3.64) in the Third Cycle
and Graded as Category–I University by MHRD-UGC]

KARAIKUDI – 630 003



DIRECTORATE OF DISTANCE EDUCATION

INTERNET AND JAVA PROGRAMMING
Master of Computer Applications